

The **keyreader** Package^{☆,★}

A robust interface to **xkeyval** package

© Ahmed Musa 2010-2012

Ahmed Musa¹

17th January 2012

Summary The **keyreader** package provides robustness and some extensions to the **xkeyval** package. It preserves braces in key values throughout parsing and saves estate when defining keys. Also, the command `\krdsetkeys` allows unbalanced conditionals to be parsed as values of keys. Furthermore, when the command `\krdddefinekeys` is used, keys are initialized as soon as they are defined, and, unlike in the **xkeyval** package, admissible alternate values of choice keys can have individual callbacks. The looping macros of the **xkeyval** package are redefined, to increase robustness. The command `\define@key` of the **xkeyval** package has had two of its subordinate commands redefined, to offset a complaint about the grabbing of the key's callback when defining keys with `\define@key`. This user manual assumes that the reader is familiar with some of the functions and user interfaces of the **xkeyval** package.

This work (i.e., all the files in the **keyreader** package bundle) may be distributed and/or modified under the conditions of the L^AT_EX Project Public License (LPPL), either version 1.3 of this license or any later version.

The LPPL maintenance status of this software is 'author-maintained.' This software is provided 'as it is,' without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

© MMXII

Contents

1	Motivation	1	2.4	Disabling keys	6
2	User commands	2	2.5	Options processing	6
2.1	Defining keys	2	3	Version history	6
2.2	Setting keys	3	Index		8
2.3	Examples	4			

1 Motivation

The **keyreader** package predated the **ltxkeys** package and was developed to make key parsing by the **xkeyval** package robust (in the sense of preserving outer braces in key values throughout

[☆] The package is available at <http://mirror.ctan.org/macros/latex/contrib/keyreader/>.

[★] This user manual corresponds to version 0.4b of the package.

¹ The University of Central Lancashire, Preston, UK. amusa22@gmail.com.

parsing and enabling the parsing of key values with unbalanced conditionals), as well as reduce the amount of typing that is required for defining several keys. To achieve robustness in key parsing, the `\setkeys` command of the `xkeyval` package has been patched and renamed `\krdsetkeys`. The `xkeyval` package's `\setkeys` remains unchanged, to avoid breaking existing codes that rely on its current form. Some other low-level commands of `xkeyval` package have been patched and renamed. The `keyreader` package provides commands for compactly defining and setting all types of key (ordinary, command, boolean, and choice). Also, the `keyreader` package introduces the concept of callbacks for the alternate/admissible values of choice keys defined via the command `\krddefinekeys`. Moreover, when `\krddefinekeys` is used, keys are automatically set/initialized as soon as they are defined. This provides default definitions for the key macros and functions. Boolean keys are initialized with a value of `false` irrespective of their default values.

The `keyreader` package has been used as a development platform for the `ltxkeys` package because the `xkeyval` package, on which the `keyreader` package is based, has been quite stable for some years, its inherent shortcomings notwithstanding. Has the user ever tried to pass to `xkeyval` package's `\setkeys` an unbalanced conditional as the value of a key? He/she will quickly be hit by the error message ‘! Incomplete `\ifx`; all text was ignored after line ...,’ or something similar. This limitation has been removed by the `keyreader` package.

2 User commands

2.1 Defining keys

The syntax for defining keys is:

New macro: `\krddefinekeys`

```
1 \krddefinekeys*[(\kprefix)]{(\kfamily)}[(\mprefix)]{(\keylist)}
```

The optional `(\kprefix)` and mandatory `(\kfamily)` have unambiguous connotations. The optional `(\mprefix)` is the macro prefix, in the parlance of the `xkeyval` package. The default values of `(\kprefix)` and `(\mprefix)` are `KRD` and `krdmp@`, respectively.

In the case of ordinary, command and boolean keys, `(\keylist)` has the syntax

Syntax of key `keylist`

```
2 {
3   <keytype-1>/<keyname-1>/<default-1>/<callback-1>;
4   <keytype-2>/<keyname-2>/<default-2>/<callback-2>;
5   ...
6   <keytype-n>/<keyname-n>/<default-n>/<callback-n>;
7 }
```

The list parser for `(\keylist)` is invariably semicolon ‘;’. Hence, if the user has semicolon ‘;’ in `(\callback)`, it has to be wrapped in curly braces, to hide it from TeX’s scanner. `(\keytype)` can be any member of the list `{ord}` (ordinary key), `cmd` (command key), `bool` (boolean key), `choice` (choice key)}.

For choice keys, `(\keylist)` has the syntax

Syntax of key `keylist`

```
8 {
9   <keytype-1>/<keyname-1>/<default-1>/<alt>/<callback-1>;
```

```

10  <keytype-2>/<keyname-2>/<default-2>/<alt>/<callback-2>;
11  ...
12  <keytype-n>/<keyname-n>/<default-n>/<alt>/<callback-n>;
13  }

```

The *alternate values* (also called *nominations* or *admissible list of user input*) `<alt>` has the syntax

Syntax of alternate list for choice keys

```

14  <value-1>.do=<callback-1>,
15  <value-2>.do=<callback-2>,
16  ...
17  <value-n>.do=<callback-n>,

```

The list parser in this case is invariably comma ‘,’.

The star (★) sign on `\krddefinekeys` is an optional suffix. If it is present, then only definable (i.e., non-existent) keys will be defined. The existence of a key depends on `<kprefix>` and `<kfamily>`, since keys are name-spaced.

Note 2.1 Choice keys defined by `\krddefinekeys` are of the starred (★) variant of choice keys (see the `xkeyval` package guide). Hence they will always convert the user input into lowercase before matching it against the alternate/admissible list of values specified at key definition time. The matching, as done by `\krdsetkeys`, is catcode-agnostic.

2.2 Setting keys

The command `\krdsetkeys` is a more robust counterpart of the `xkeyval` package’s `\setkeys`, in the sense that it preserves all outer braces in the values of keys and allows the parsing of key values with unbalanced conditionals. The new command `\krdsetkeys` has the same syntax as the original `\setkeys` of the `xkeyval` package, namely

New macro: `\krdsetkeys`

```

18  \krdsetkeys★+ [<kprefix>]{<families>}[<na>]{<(key)=<value>>}

```

As usual, the star (★) and plus sign (+) are optional suffixes. The starred (★) variant will save all undeclared keys in the list `\XKV@rm`, possibly for setting later with the command `\setrmkeys`, and will not report any unknown key as undeclared. The plus (+) variant will set the given keys in all the given families, instead of in just one family. The combination ★+ will set the listed keys in all the given families and append unknown keys to the container `\XKV@rm`. `<na>` is the list of keys that shouldn’t be set in the current run.

The command `\krdsetkeys` isn’t exactly `xkeyval` package’s `\setkeys`, since the former is more robust and avoids the selective sanitization of `<key>=<value>` list that is done by `\setkeys`. The macro `\krdsetkeys` ‘normalizes’ the `<key>=<value>` or comma-separated key list. Therefore, users of the `keyreader` package should always call the command `\krdsetkeys` instead of `\setkeys`. Both have the same user interface. The `\setkeys` command of the `xkeyval` package’s remains unchanged.

The `xkeyval` package’s command `\setrmkeys`, which sets ‘remaining keys,’ has also been modified to `\krdsetrmkeys`, while keeping `\setrmkeys` unchanged. Users of the `keyreader` package should use `\krdsetrmkeys` in place of `\setrmkeys`.

2.3 Examples

Examples 2.1: \krddefinekeys, \krdsetkeys

```

19 \krddefinekeys*[KV]{fam}[pnt@]{%
20   % '#1' throughout here refers to the user input for the key.
21   ord/keya/{black}/\def\xx##1{#1##1};
22   cmd/keyb/\@firsrtofone/\def\y##1{#1##1};
23   bool/keyc/true/\def\z##1{#1##1};
24   choice/keyd/center/
25     center.do=\def\vcp@align{center}\def\w##1{#1##1},
26     left.do=\def\vcp@align{flushleft},
27     right.do=\def\vcp@align{flushright},
28     justified.do=\def\vcp@align{relax}/
29   \ifkrddef\else
30     \def\xa##1{#1##1}
31   \fi;
32   ord/keye/{keye-default}/\def\y##1{#1##1}
33 }
34 \krdsetkeys[KV]{fam}[keyb]{keya={green},keyb=\@iden,keyc=false,keyd=left}
35 % Setting keys with values having unbalanced conditionals:
36 \krdsetkeys[KV]{fam}{keye={\iffalse keye-value}}
```

The braces around 'green,' the value of `keya`, will be preserved throughout parsing. It should be remembered that keys are automatically set as soon as they are defined by `\krddefinekeys`. The boolean `\ifkrddef` is true when `\krddefinekeys` is defining keys, and false otherwise. The essence of it is that since keys are set as soon as they are defined by `\krddefinekeys`, some actions should not be executed at this time, until the keys are being set by the user.

Using the keys defined in the above example, let us make comma ',' and comma '=' active and see how the `keyreader` package will deal with them.

Example 2.2: Active comma and equals sign

```

37 % Make comma ',' and equal '=' active to test the list normalization
38 % scheme of 'keyreader' package:
39 \begingroup
40 \catcode'\,=13
41 \catcode'\==13
42 \gdef\keylista{{fam,famb}[keyb , keyc]{keya = {green} , keyb = \@iden ,
43   keyc = false , keyd = left, keye = somevalue}}
44 \gdef\keylistb{\krdsetrmkeys*+[KV]{fam,famb}}
45 \endgroup
46 \def\reserved@a{\krdsetkeys*+[KV]}
47 \expandafter\reserved@a\keylista
48 \keylistb
```

The output of the following example is shown in [Figure 1](#):

Example 2.3

```

49 \documentclass{article}
50 \usepackage{keyreader}
51 \usepackage{xcolor}
52 \makeatletter
```

```

53 \newdimen\shadowsize
54 \krddefinekeys*[KV]{ebox}[mp@]{%
55   bool/frame/true;
56   bool/shadow/true;
57   cmd/framecolor/black;
58   cmd/shadecolor/white;
59   cmd/shadowcolor/gray;
60   cmd/framesize/.4pt;
61   cmd/boxsize/.1\columnwidth;
62   cmd/shadowsize/1pt;
63   choice/align/center/
64     center.do=\let\mp@alignright\hfil\let\mp@alignleft\hfil,
65     right.do=\let\mp@alignright\hfill\let\mp@alignleft\relax,
66     left.do=\let\mp@alignright\relax\let\mp@alignleft\hfill,
67     justified.do=\let\mp@alignright\relax\let\mp@alignleft\relax
68   /
69   \def\userinput{#1};
70 }
71 \savekeys[KV]{ebox}{frame,framecolor,framesize}
72 % 'Preset keys' have no 'tail keys':
73 \krdpresetkeys[KV]{ebox}{%
74   frame,framecolor=black,framesize=0.5pt,boxsize,align
75 }
76 % 'Postset keys' have no 'head keys':
77 \krdpostsetkeys[KV]{ebox}{%
78   shadow=\usevalue{frame},shadowcolor=\usevalue{framecolor}!40,
79   shadowsize=\usevalue{framesize}*4
80 }
81 \newcommand*\ebox[2][ ]{%
82   \setkeys[KV]{ebox}{#1}%
83   % What happens if we use the following, instead of the above \setkeys?
84   % Preset and postset keys wouldn't be set when '#1' is empty:
85   % \krdifblank{#1}{-}{\setkeys[KV]{ebox}{#1}}
86   \begingroup
87   \ifmp@frame
88     \fboxrule=\dimexpr\mp@framesize\relax
89   \else
90     \fboxrule=0pt
91   \fi
92   \ifmp@shadow
93     \shadowsize=\dimexpr\mp@shadowsize\relax
94   \else
95     \shadowsize=0pt
96   \fi
97   \setbox0=\hbox{%
98     \fcolorbox{\mp@framecolor}{\mp@shadecolor}{%
99       \hbox to\mp@boxsize{%
100         \mp@alignright #2\mp@alignleft
101       }%
102     }%
103   }%
104   \hskip\shadowsize
105   \color{\mp@shadowcolor}%

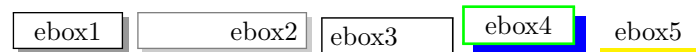
```

```

106 \rule[-\dp0]{\wd0}{\the\dimexpr\ht0+\dp0\relax}%
107 \llap{\raisebox{\shadowsize}{\box0\hskip\shadowsize}}%
108 \endgroup
109 }
110 \makeatother

111 \begin{document}
112 \ebox{ebox1}
113 \ebox[framecolor=gray,boxsize=2cm,align=right]{ebox2}
114 \ebox[shadow=false,boxsize=1.5cm,align=left]{ebox3}
115 \ebox[framesize=1pt,framecolor=green,shadowcolor=blue]{ebox4}
116 \ebox[frame=false,shadow,shadowcolor=yellow,framesize=.5pt]{ebox5}
117 \end{document}

```

Figure 1: Output of [example 2.3](#)

2.4 Disabling keys

The command `\krddisablekeys` has the same use syntax as the `\disable@keys` command of the `xkeyval` package, but will issue an error (instead of a warning) when an attempt is made to set a disabled key.

2.5 Options processing

The commands `\krdDeclareOption`, `\krdExecuteOptions` and `\krdProcessOptions` are aliases for `\DeclareOptionX`, `\ExecuteOptionsX` and `\ProcessOptionsX` of the `xkeyval` package.

3 Version history

The following change history highlights significant changes that affect user utilities and interfaces; changes of technical nature are not documented in this section.

Version 0.4b [2012/01/14]

The command `\krdsetkeys` can now parse key values with unbalanced conditionals.

Version 0.4a [2011/12/23]

The key list in `\krddefinekeys` shouldn't have been normalized with respect to forward slash (/).

Version 0.4 [2011/12/20]

Several of the former functions of the package have been transferred to the `ltxkeys` package with even more robustness. The package now provides mainly a compact and robust interface to the features of the `xkeyval` package.

Version 0.3 [2011/03/26]

Bug fix.

Version 0.2 [2011/02/25]

The interface for defining new keys now accepts conditionals in key macros/functions.

A mechanism is provided for automatic setting up and execution of key functions with default key values.

Version 0.1 [2010/01/10]

First public release.

INDEX

Index numbers refer to page numbers.

A		
active comma and equals sign.....	4	
D		
<code>\DeclareOptionX</code>	6	
<code>\disable@keys</code>	6	
disabling keys.....	6	
E		
examples.....	3	
<code>\ExecuteOptionsX</code>	6	
K		
<code>\krdDeclareOption</code>	6	
<code>\krddefinekeys</code>	2	
<code>\krddisablekeys</code>	6	
<code>\krdExecuteOptions</code>	6	
<code>\krdpostsetkeys</code>	6	
<code>\krdpresetkeys</code>	6	
<code>\krdProcessOptions</code>	6	
		<code>\krdsetkeys</code>3
		<code>\krdsetrmkeys</code>3
		O
		options processing.....6
		P
		Packages.....
		<code>keyreader</code>1–4
		<code>ltxkeys</code>1, 2, 6
		<code>xkeyval</code>1–3, 6
		<code>\ProcessOptionsX</code>6
		S
		<code>\setkeys</code>3
		<code>\setrmkeys</code>3
		U
		user commands.....2