The keyreader Package[☆,★]

A robust interface to xkeyval package

Ahmed Musa[1]

24th December 2011

Summary The keyreader package provides robustness and some extensions to the xkeyval package. It preserves braces in key values and saves estate when defining keys. Also, when the command \krddefinekeys 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. This user manual assumes that the reader is familiar with some of the functions and user interfaces of the xkeyval package.

Contents

1 Motivation 2
2 User commands 2
3 Examples 3
4 Disabling keys 4
5 Options processing 4
6 Version history 4
Index 6

[☆] The package is available at http://mirror.ctan.org/macros/latex/contrib/keyreader/.
[★] This user manual corresponds to version 0.4a of the package.
[1] The University of Central Lancashire, Preston, UK. amusa22@gmail.com.

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 parsing), 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 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 not withstanding. Because the keyreader package is based on the xkeyval package, it inherits some of the limitations of the xkeyval package. Has the user ever tried to pass to xkeyval package's \setkeys an unbalanced conditional (invloving \iftrue, \iffalse, \ifx, or \fi) 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. The same limitation applies to the keyreader package. In this respect, the ltxkeys package is more robust.

2 User commands

The syntax for defining new 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   ⟨keytype1⟩/⟨keyname1⟩/⟨default1⟩/⟨callback1⟩;
4   ⟨keytype2⟩/⟨keyname2⟩/⟨default2⟩/⟨callback2⟩;
5   etc.
6 }
```

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

```
7 {
8   ⟨keytype1⟩/⟨keyname1⟩/⟨default1⟩/⟨alt⟩/⟨callback1⟩;
```

```
  ⟨keytype2⟩/⟨keyname2⟩/⟨default2⟩/⟨alt⟩/⟨callback2⟩;
  etc.
}
```

The alternate (admissible list of) values ⟨alt⟩ has the syntax

Syntax of alternate list for choice keys

```
⟨value1⟩.code=⟨callback1⟩,
⟨value1⟩.code=⟨callback1⟩,
etc.
```

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

The star (⋆) is an optional prefix. 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.

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. The new command \krdsetkeys has the same syntax as the original \setkeys of the xkeyval package, namely

Macro: \setkeys

```
\krdsetkeys*+[⟨kprefix⟩]{⟨families⟩}[⟨na⟩]{⟨⟨key⟩=⟨value⟩ pairs⟩}
```

As usual, the star (⋆) and plus sign (+) are optional prefixes. 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.

Since a package might redefine \setkeys, we also saved the patched \setkeys in \krdsetkeys. Indeed, \krdsetkeys isn't exactly \setkeys, since the former avoids the selective sanitization of ⟨key⟩=⟨value⟩ list that is done by \setkeys. Instead \krdsetkeys ‘normalizes’ the ⟨key⟩=⟨value⟩ or comma-separated list. Therefore, users of the keyreader package should always call the command \krdsetkeys instead of \setkeys. Both have the same user interface.

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

3 Examples

Examples: \krddefinekeys, \krdsetkeys

```
\krddefinekeys*[KV]{fam}[pnt@]{%
  % ‘#1’ here refers to the user input for the key.
  ord/keya/{black}/\def\xx##1{#1##1};
  cmd/keyb/\@fisrtofone/\def\y##1{#1##1};
  bool/keyc/true/\def\z##1{#1##1};
  choice/keyd/center/
    center.code=\def\my@align{center}\def\w##1{#1##1},
    left.code=\def\my@align{flushleft},
    right.code=\def\my@align{flushright},
```

```
    justified.code=\def\my@align{relax}%
    /\def\xa##1{#1##1};
}
\krdsetkeys[KV]{fam}[keyb]{keya={green},keyb=\@iden,keyc=false,keyd=left}
```

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.

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: Active comma and equal sign

```
% Make comma ',' and equal '=' active to test the list normalization
% scheme of 'keyreader' pacakge:
\begingroup
\catcode`\,=13
\catcode`\==13
\gdef\keylista{{fam,famb}[keyb , keyc]{keya = {green} , keyb = \@iden ,
  keyc = false , keyd = left, keye = somevalue}}
\gdef\keylistb{\krdsetrmkeys*+[KV]{fam,famb}}
\endgroup
\def\reserved@a{\krdsetkeys*+[KV]}
\expandafter\reserved@a\keylista
\keylistb
```

4 Disabling keys

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

5 Options processing

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

6 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 0.4a [2011/12/23]

Bug fix: 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 equal sign . . . 4

D

\DeclareOptionX . . . 4
\disable@keys . . . 4
disabling keys . . . 4

E

examples . . . 3
\ExecuteOptionsX . . . 4

K

\krdDeclareOption . . . 4
\krddefinekeys . . . 2, 4
\krddisablekeys . . . 4
\krdExecuteOptions . . . 4
\krdProcessOptions . . . 4
\krdsetkeys . . . 4
\krdsetrmkeys . . . 3

O

options processing . . . 4

P

Packages . . .
keyreader . . . 1–4
ltxkeys . . . 2, 4
xkeyval . . . 1–4
\ProcessOptionsX . . . 4

S

\setkeys . . . 4
\setrmkeys . . . 3

U

user commands . . . 2