The keyreader Package*

Ahmed Musa
a.musa@rocketmail.com

January 10, 2010

Contents

1 Motivation

The elaborate and powerful xkeyval package introduced \define@cmdkeys and \define@boolkeys, which permit the setting of multiple command keys and boolean keys, but in each category those keys must have the same default value and no key macro/function. This package seeks to remove these restrictions, so that multiple keys of all categories (ordinary keys, command keys, boolean keys, and choice keys) can be set at one go and those keys can have different default values and functions. This greatly minimizes tokens, as hundreds of keys can, in principle, be issued simultaneously by one command.

Also, the xkeyval package doesn't have the notion of complementary keys, which can be handy in the case of boolean keys. The present package introduces this concept and additionally permits the submission of individual/different key macros to the complementary keys.

*Version 0.1.

2 Usage

The package can be loaded in style files by

```
\RequirePackage[options]{keyreader}
```

and in document files via

```
\usepackage[options]{keyreader}
```

where the options and their default values are

```
parser=;, macroprefix=mp@, keyprefix=KV, keyfamily=myfamily.
```

The `parser` is the separator between the keys in the key list to be defined in one go (see examples in section 3). All these options can be set dynamically by using the `\krsetup` macro:

```
\krsetup{parser=;,macroprefix=mp@,keyprefix=KV,
  keyfamily=myfamily}.
```

The main user interface is the `\define@keylist` macro, whose syntax is

```
\define@keylist{<key type/id>, <key>, <key value>,
  <key macro/function>; <another set of key specifiers>; etc}
```

There are four key types: 1 (ordinary key), 2 (command key), 3 (boolean key), 4 (choice key). The key and its attributes are separated by commas; they constitute one `object`. The objects are separated by the `parser`, which is the semicolon in the above example.

If the key list is available in a macro, say,

```
\def\keylist{<key type/id>, <key>, <key value>,
  <key macro/function>; <another set of key specifiers>; etc},
```

then the keys can be defined by

```
\cmddefine@keylist\keylist.
```

`\cmddefine@keylist` takes a macro as argument, while `\define@keylist` accepts a key list.

The \kr@altlist macro is needed for choice keys; it lists the alternate admissible values for a choice key and thus can't be empty when a choice key is being defined or set. It has to be defined or redefined each time a choice key is being defined, if its contents (ie, those of \kr@altlist) change from one choice key to the next one being defined. This, unfortunately, implies the restriction that no two choice keys can appear in one \define@keylist statement. This restriction can be easily lifted but at the cost of some complication and extra requirement for the package user. So far this hasn't been a severe limitation (to me, anyway).

3 Examples

Suppose we wish to define a set of keys {color,angle,scale,align}. The keys color, angle and scale will de defined using command keys, while the key align will be defined by choice keys. Assume that the align key can only assume the values {center,right,left,justified}, where the first three values would further imply \centering, \flushright, and \flushleft, respectively. Moreover, we assume that the key scale will be associated with a macro called \mydo, where \do is assumed defined elsewhere. The keys color and angle aren't associated with macros. Then we can go:

```
\def\kr@altlist{center,right,left,justified}

\def\f@align{%
  \ifcase\kr@nr\relax
    \def\kr@align{\centering}%
  \or
    \def\kr@align{\flushright}%
  \or
    \def\kr@align{\flushleft}%
  \or
    \let\kr@align\relax\fi
}

\define@keylist{2,color,gray!25,;2,angle,45,;
  2,scale,1,\def\mydo##1{\do ##1};4,align,center,\f@align;
  \stopread;3,mybool,true,}.
```

The \kr@nr macro is a bin parameter defined by the keyreader package in accordance with the instructions in the xkeyval guide. It refreshes with the choice key. See the xkeyval guide for further details.

Instead of defining the macro \f@align before hand, we can submit its replacement text directly to the macro \define@keylist, but, because \f@align

contains a conditional, some care is needed in doing so (see section 6). Once the key `align` has been defined, the macro `\f@align` can be reused—perhaps to define other keys—even before the key `align` is set. This is because it isn't `\f@align` that is used in defining the key `align` but its "meaning" (i.e., the meaning of `\f@align`). In this way, the user can economize on tokens. The same applies to all the macros that may be used in defining keys.

Note the `\stopread` command inserted above. Because of it, the key `mybool` will not be read and defined; the rest (i.e., `color`, `angle`, `scale` and `align`) will be read and defined. All the entries for `mybool` will instead be saved in the macro `\kr@remainder`.

Hundreds of keys can be defined efficiently in this way, using very few tokens.

As another example, we consider the following page setup keys:

```
\def\kr@altlist{center,right,left,justified} % reused

\define@keylist{%
  3,boolvar,true,;1,paperheight,\paperheight,;
  1,paperwidth,\paperwidth,\f@paperwidth;
  2,textheight,\textheight,\f@textheight;
  2,textwidth,\textwidth,\f@textwidth;
  1,evensidemargin,\evensidemargin,;
  4,align,center,\f@align;
  2,testdim,2cm,\long\def\f@testdim##1{A test dimension ##1
    \par\bigskip}%
}
```

which have the following trivial key macros:

```
\def\f@textwidth{\AtBeginDocument{\wlog{‘textwidth’ %
  is \mp@textwidth}}}

\def\f@textheight{%
  \ifx\@empty\mp@textheight
    \wlog{‘textheight’ value empty}%
  \else
    \wlog{‘textheight’ value not empty}%
  \fi
}

\def\f@paperwidth{\wlog{‘paperwidth’ was defined as %
  ordinary key.}}
\newcommand\f@align{%
  \ifcase\kr@nr\relax
```

```
    \def\mp@align{\centering}%
  \or
    \def\mp@align{\flushright}%
  \or
    \def\mp@align{\flushleft}%
  \or
    \let\mp@align\relax
  \fi
}
```

Again, once the keys have been defined, these macros can be reused.

Since the keys have been defined, they can now be set. In the following, we set only two of the keys:

```
\setkeys[KV]{myfamily}{align=right,testdim=3cm}
```

The macro `\mp@align` holds the value `\flushright`, while

```
\KV@myfamily@testdim
```

holds the macros:

```
\def\mp@testdim{#1}
\long\def\f@testdim##1{A test dimension##1\par\bigskip},
```

where `#1` is the value submitted for the key `testdim`. Try `\show\mp@align`, `\show\KV@myfamily@testdim`, and `\show\f@testdim` to confirm the above assertions.

The rest of the defined keys can now be set as follows:

```
\setkeys[KV]{myfamily}{boolvar=true,paperheight,paperwidth,
  textheight,textwidth=6cm}
```

Try `\show\ifmp@boolvar` to confirm that `boolvar` is now `true`; it is originally set as `false`. The macro `\KV@myfamily@paperwidth` holds the function `\f@paperwidth`; `\mp@textheight` holds the value submitted to key `textheight` at any instance of `\setkeys`. By the above `\setkeys`, only the default values of `paperheight`, `paperwidth`, and `textheight` are presently available.

3.1 Demonstrating the effect of limiting \textwidth

After defining and setting the keys above, the following tokens can be used in a source file to demonstrate the effect of setting the page setup keys shown above:

```
\begin{center}
  \begin{minipage}{\mp@textwidth}
    \lipsum[1] % from the lipsum package
  \end{minipage}
\end{center}

\lipsum[1]
```

4 Complementary boolean keys

The syntax of complementary boolean keys is

```
\define@comp@boolkeys[<key-prefix>]{<family>}[<macro-prefix]
  {<primary boolean>}[<default value for primary boolean>]
  {<secondary boolean>}{<func for primary boolean>}
  {<func for secondary boolean>}.
```

When the user doesn't supply the `key-prefix` and/or `macro-prefix`, the package will use `KV` and `mp@`, respectively. When one boolean (primary or secondary) is true, the other is automatically set false. Infinite loops, which are possible in back-linked key settings, are avoided in the `keyreader` package.

As an example, we define below two complementary keys `draft` and `final` with different key macros:

```
\define@comp@boolkeys[KV]{myfamily}[kr@]{draft}[true]{final}%
  {%
    \def\gobble##1{}%
  }{%
    \def\notgobble##1{##1}%
  }.
```

The key prefix (default `KV`), macro prefix (default `mp@`), and key macros (no default) can be empty:

```
\define@comp@boolkeys{myfamily}{draft}[true]{final}{}{}.
```

The defined complementary keys `draft` and `final` can now be set as follows:

```
\setkeys[KV]{myfamily}{draft=true}

\setkeys[KV]{myfamily}{final=true}
```

The second statement above reverses the boolean `draft` to `false`, which had been set in the first statement to `true`. There is no meaning to the following:

```
\setkeys[KV]{myfamily}{draft=true,final=true}.
```

As an exercise, in a document file, after defining and setting the above keys, one can issue the command `\krsetup{draft=true}` and do

```
\show\ifkr@draft \show\ifkr@final \show\gobble
\show\KV@myfamily@draft.
```

These statements assume the macro prefix to be `kr@`. Next, we may set

```
\krsetup{final=true}
```

and do

```
\show\ifkr@final \show\ifkr@draft \show\notgobble.
```

5 Input error

Both boolean and choice keys issue error messages if the input is not valid, i.e., not in the list of admissible values. The default input error is defined by `\kr@inputerr` macro to be

```
<Erroneous value ‘#1’ for key ‘#2’>.
```

`\kr@inputerr` can be redefined by the user. It takes two arguments (i.e., value and key).

6 Conditionals in key macros

The TEX conditional primitives \if and \fi cannot appear in the key macro when \define@keylist is being invoked. The reason can be traced to the discussion on page 211 of the TEXBook. Key macros/functions involving conditional operations such as

```
\ifmp@boolone \do \fi
```

can be submitted to \define@keylist via macros, as seen above. We give more examples below.

Suppose we want to submit the following:

```
\define@keylist{3,boolone,true,\ifmp@boolone \do \fi}.
```

The presence of \if... and \fi in the argument will trigger an error when TEX is scanning or skipping tokens (see the TEXBook). Neither \protect nor \noexpand is helpful here. One solution is to first define

```
\def\f@boolone{\ifmp@boolone \do \fi}
```

and then do

```
\define@keylist{3,boolone,true,\f@boolone},
```

which will execute \f@boolone when the key boolone is set. Once the key boolone has been defined by the above statement, the function \f@boolone may be redefined and reused many times, any time, even before the setting of the key boolone. It isn't the function \f@boolone that is used in defining the key boolone, but the meaning of \f@boolone.

As another example, we may do

```
\def\f@bool{\ifmp@bool\def\do####1{%
  \def####1########1{\expandafter\expandafter\expandafter\in@
  \expandafter\expandafter\expandafter{\expandafter####1
  \expandafter}\expandafter{########1}}}\fi}

\define@keylist{3,bool,true,\f@bool}.
```

"Toggles," introduced in the etoolbox package, can also be used to circumvent the problem of matching \if and \fi in difficult circumstances, since toggles aren't TEX primitives, but the xkeyval doesn't as yet have a mechanism for

defining and setting toggle keys. However, this shouldn't be difficult to implement in the future.

7 Disabling keys

The keyreader package has modified the definition of \disable@keys from the xkeyval package to allow for bespoke warnings and error messages. The new command is \krdisable@keys; the use syntax remains the same as that of \disable@keys:

```
\krdisable@keys[<key prefix>]{<key family>}{<comma %
  separated list of keys to disable>}.
```

Any attempt to subsequently set a disabled key will prompt the following error message. (The xkeyval package issues a warning in this case.) The error message can be modified by the user, but the "names" \disabledkeyerr and \disabledkey should be retained.

```
\def\disabledkeyerr{%
  \PackageError{keyreader}{%
    Key `\disabledkey' has been disabled.\MessageBreak
    You can't set or reset it at this late stage.\MessageBreak
    You should have set it earlier in the\MessageBreak
    \string\documentclass\space or \string\usepackage
  }{\@ehc}%
}
```

If the user attempts to disable an undefined key, the xkeyval package issues a fatal error; the keyreader package, on the other hand, issues a warning in the transcript .log file, since the situation isn't fatal to the outcome.