The keyval package*

David Carlisle

1999/03/16

Abstract

A LaTeX package implementing a system allowing the setting of parameters (or 'named arguments' with a ⟨*key*⟩ = ⟨*value*⟩ syntax.

Eg: `\foo[height=3in, shadow = true ]{bar}`

This package implements a system of defining and using sets of parameters, which are set using the syntax ⟨*key*⟩=⟨*value*⟩.

For each keyword in such a set, there exists a function which is called whenever the parameter appears in a parameter list. For instance if the set `dpc` is to have the keyword `scale` then I would define.

```
\define@key{dpc}{scale}{scale ({\tt\string#1})\\}
```

The first argument of `\define@key` is the set of keywords being used, the second is the keyword, and the third is the function to call. This function will be given as `#1` the ⟨*value*⟩ specified by the user.

Normally it is an error to omit the '=⟨*value*⟩' however if an optional ⟨*value*⟩ is supplied when the keyword is defined, then just the keyword need be supplied.

```
\define@key{dpc}{clip}[true]{...}
```

For '`clip`' you can go '`clip = true`' or '`clip = false`' or just '`clip`', which is the same as '`clip = true`'

To use these keywords, just call '`\setkeys`' with a comma separated list of settings, each of the form ⟨*key*⟩=⟨*value*⟩, or just ⟨*key*⟩. Any white space around the '`=`' and '`,`' is ignored.

As the ⟨*key*⟩ is passed as a macro argument, if it consists entirely of a `{ }` group, the outer braces are stripped off. Thus `,key=foo,` and `,key={foo},` are equivalent. This fact enables one to 'hide' any commas or equals signs that must appear in the value. i.e. in `foo={1,2,3},bar=4`, `foo` gets the value `1,2,3`, the comma after `1` does not terminate the keyval pair, as it is 'hidden' by the braces.

Empty entries, with nothing between the commas, are silently ignored. This means that it is not an error to have a comma after the last term, or before the first.

1 Example

We may extend the examples above to give a 'fake' graphics inclusion macro, with a syntax similar to that used in the psfig macros.

*This file has version number v1.13, last revised 1999/03/16.

`\dpcgraphics` has one optional argument which is passed through `\setkeys`, and one mandatory argument, the filename. It actually just typesets its arguments, for demonstration.

The declared keys are: `scale`, `height`, `width`, `bb`, and `clip`. Except for the last, they must all be given a value if used.

Note how in the following, any white space arround `=` or `,` is ignored, as are the 'empty' arguments caused by extra commas. Note also that each macro receives *exactly* the tokens that you specify as arguments, no premature expansion is done.

```
\def\dpcgraphics{\@ifnextchar[\@dpcgraphics{\@dpcgraphics[]}}
\def\@dpcgraphics[#1]#2{{\setkeys{dpc}{#1}INPUT: #2}}

\define@key{dpc}{scale}{scale ({\tt\string#1\relax})\\}
\define@key{dpc}{clip}[true]{clip ({\tt\string#1\relax})\\}
```

```
\def\scalemacro{9}
\dpcgraphics
[ height =4in, ,
  width = 3in,
  scale = \scalemacro,
  bb = 20 20 300 400 ,
  clip,
 ]{aaa}
```

height (`4in`)
width (`3in`)
scale (`\scalemacro`)
bounding box (`20 20 300 400`)
clip (`true`)
INPUT: aaa

2 The Internal Interface

A declaration of the form:
`\define@key{family}{key}{...}`
Defines a macro `\KV@prefix@key` with one argument. When used in a keyval list, the macro receives the value as its argument.

A declaration of the form:
`\define@key{family}{key}[default]{...}`
Defines a macro `\KV@family@key` as above, however it also defines the macro `\KV@family@key@default` as a macro with no arguments, and definition `\KV@family@key{default}`.

Thus if macros are defined using `\define@key`, the use of a key with no value `...,foo,...` is always equivalent to the use of the key with some value, `...,foo=default,...`. However a package writer may wish that the 'default' behaviour for some key is not directly equivalent to using that key with a value. (In particular, as pointed out to me by Timothy Van Zandt, you may wish to omit error checking on the default value as you know it is correct.) In these cases one simply needs to define the two macros `\KV@`⟨*family*⟩`@key` and `\KV@`⟨*family*⟩`@key@default` directly using `\def` (or `\newcommand`). I do not supply a user interface for this type of definition, but it is supported in the sense that I will try to ensure that any future upgrades of this package do not break styles making use of these 'low level' definitions.

3 The Macros

From version 1.05, all 'internal' macros associated to keys have names of the form: `\KV@`⟨*family*⟩`@`⟨*key*⟩ or `\KV@`⟨*family*⟩`@`⟨*key*⟩`@`⟨*default*⟩

```
⟨*package⟩
```

`\setkeys` The top level macro. `#2` should be a comma separated values of the form ⟨*key*⟩ `=` ⟨*value*⟩ or just simply ⟨*key*⟩. The macro associated with this key in the 'family' `#1` is called with argument ⟨*value*⟩. The second form is only allowed if the key was declared with a default value.

```
\def\setkeys#1#2{%
```

Save the 'family' for later. Then begin acting on the comma separated list.

```
  \def\KV@prefix{KV@#1@}%
  \let\@tempc\relax
  \KV@do#2,\relax,}
```

`\KV@do` Iterate down the list of comma separated argument pairs.

```
\def\KV@do#1,{%
 \ifx\relax#1\empty\else
  \KV@split#1==\relax
  \expandafter\KV@do\fi}
```

`\KV@split` Split up the keyword and value, and call the appropriate command. This macro was slightly reorganised for version 1.04, after some suggestions from Timothy Van Zandt.

```
\def\KV@split#1=#2=#3\relax{%
  \KV@@sp@def\@tempa{#1}%
  \ifx\@tempa\@empty\else
    \expandafter\let\expandafter\@tempc
      \csname\KV@prefix\@tempa\endcsname
    \ifx\@tempc\relax
⟨plain⟩        \KV@err
⟨!plain⟩        \KV@errx
       {\@tempa\space undefined}%
    \else
      \ifx\@empty#3\@empty
        \KV@default
      \else
        \KV@@sp@def\@tempb{#2}%
        \expandafter\@tempc\expandafter{\@tempb}\relax
      \fi
    \fi
  \fi}
```

`\KV@default` Run the default code, or raise an error.

```
\def\KV@default{%
  \expandafter\let\expandafter\@tempb
    \csname\KV@prefix\@tempa @default\endcsname
  \ifx\@tempb\relax
    \KV@err{No value specified for \@tempa}%
  \else
    \@tempb\relax
  \fi}
```

\KV@err Error messages.

```
⟨plain⟩\def\KV@err#1{\errmessage{key-val: #1}}
⟨*!plain⟩
\DeclareOption{unknownkeysallowed}{%
  \def\KV@errx#1{\PackageInfo{keyval}{#1}}}
\DeclareOption{unknownkeyserror}{%
  \def\KV@errx#1{\PackageError{keyval}{#1}\@ehc}}
\ExecuteOptions{unknownkeyserror}
\let\KV@err\KV@errx
\ProcessOptions
⟨/!plain⟩
```

\KV@@sp@def \KV@@sp@b \KV@@sp@c \KV@@sp@d

`\KV@@sp@def`⟨*cmd*⟩⟨*token list*⟩ is like `\def`, except that a space token at the beginning or end of ⟨*token list*⟩ is removed before making the assignment. ⟨*token list*⟩ may not contain the token `\@nil`, unless it is within a brace group. The names of these commands were changed at version 1.05 to ensure that they do not clash with 'internal' macros in a key family 'sp'.

Since v1.10, `#` may appear in the second argument without it needing to be doubled as `##`. Also earlier versions would drop any initial brace group, so `{abc}d` would incorrectly be treated as `abcd`. The current version only removes brace groups that surround the entire value, so `{abcd}` *is* treated correctly as `abcd`. In fact, two levels of bracing are removed, if you require the entire argument to be a single brace group, you must use `{{{abcd}}}`.

```
\def\@tempa#1{%
```

```
\def\KV@@sp@def##1##2{%
  \futurelet\KV@tempa\KV@@sp@d##2\@nil\@nil#1\@nil\relax##1}%
```

Early release removed initial space by having an 'extra' argument in `\KV@@sp@b` but that removed too many braces, so now make `\KV@@sp@b` explicitly remove a single space token. That unfortunately means we need the new `\KV@@sp@d` command to add a space token if one was not there before.

```
\def\KV@@sp@d{%
  \ifx\KV@tempa\@sptoken
    \expandafter\KV@@sp@b
  \else
    \expandafter\KV@@sp@b\expandafter#1%
  \fi}%
```

```
\def\KV@@sp@b#1##1 \@nil{\KV@@sp@c##1}%
```

```
⟨plain⟩\def\@sptoken{#1}%
```

Make the above definitions, inserting the space token where needed.

```
  }
\@tempa{ }
```

```
\def\KV@@sp@c#1\@nil#2\relax#3{\KV@toks@{#1}\edef#3{\the\KV@toks@}}
```

\KV@toks@ Macro register used above to prevent `#` doubling. Avoid uding one of the normal scratch registers, as this code is not in a local group.

```
\newtoks\KV@toks@
```

\define@key Define the command associated to the key `#2` in the family `#1`. First looks for a default argument (the default value for the key)

```
\def\define@key#1#2{%
  \@ifnextchar[{\KV@def{#1}{#2}}{\@namedef{KV@#1@#2}####1}}
```

`\KV@def` Make the definitions of the command, and the default value.

```
\def\KV@def#1#2[#3]{%
  \@namedef{KV@#1@#2@default\expandafter}\expandafter
    {\csname KV@#1@#2\endcsname{#3}}%
  \@namedef{KV@#1@#2}##1}
```

⟨/package⟩