define expandable \langle key\rangle=\langle value\rangle macros using expkv

Jonathan P. Spratte*
2020-04-04 v0.1

Abstract
expkvcs provides two small interfaces to define expandable \langle key\rangle=\langle value\rangle macros utilizing expkv. It therefore lowers the entrance boundary to expandable \langle key\rangle=\langle value\rangle macros. The stylised name is expkvcs but the files use expkv-cs, this is due to CTAN-rules which don’t allow | in package names since that is the pipe symbol in *nix shells.

Contents
1 Documentation 2
1.1 Define Macros and Primary Keys 2
1.1.1 Primary Keys 2
1.1.2 Split 3
1.1.3 Hash 3
1.2 Secondary Keys 4
1.2.1 p-type Prefixes 4
1.2.2 t-type Prefixes 4
1.3 Example 5
1.4 Useless Macros 7
1.5 Bugs 7
1.6 License 7

2 Implementation 8
2.1 The \LaTeX{} Package 8
2.2 The Generic Code 8
2.2.1 Secondary Key Types 16
2.2.2 Helper Macros 18
2.2.3 Assertions 18
2.2.4 Messages 18

Index 20

*jspratte@yahoo.de
1 Documentation

The \texttt{exPkv} package enables the new possibility of creating \langle \texttt{key} \rangle = \langle \texttt{value} \rangle macros which are fully expandable. The creation of such macros is however cumbersome for the average user. \texttt{exPkv\_cs} tries to step in here. It provides interfaces to define \langle \texttt{key} \rangle = \langle \texttt{value} \rangle macros without worrying too much about the implementation. There are two different approaches supported by this package. The first is splitting the keys up into individual arguments, the second is providing all the keys as a single argument to the underlying macro and getting an individual \langle \texttt{value} \rangle by using a hash. Well, actually there is no real hash, just some markers which are parsed, but this shouldn’t be apparent to the user, the behaviour matches that of a hash-table.

In addition to these two methods of defining a macro with primary keys a way to define secondary keys, which can reference the primary ones, is provided. These secondary keys don’t correspond to an argument or an entry in the hash table directly but might come in handy for the average use case. Each macro has its own set of primary and secondary keys.

A word of advice you should consider: If your macro doesn’t have to be expandable (and often it doesn’t) don’t use \texttt{exPkv\_cs}. The interface has some overhead (though it still can be considered fast) and the approach has its limits. If you don’t need to be expandable, you should consider either defining your keys manually using \texttt{exPkv} or using \texttt{exPkv\_def} for convenience. Or you resort to another \langle \texttt{key} \rangle = \langle \texttt{value} \rangle interface. \texttt{exPkv\_cs} is usable as generic code and as a \LaTeX{} package. It’ll automatically load \texttt{exPkv} in the same mode as well. To use it, just use one of

\begin{verbatim}
\usepackage{expkv-cs} % LaTeX
\input expkv-cs % plainTeX
\end{verbatim}

1.1 Define Macros and Primary Keys

All macros defined with \texttt{exPkv\_cs} have to be previously undefined or have the \texttt{\meaning of \relax}. This is necessary as there is no way to undefine keys once they are set up (neither \texttt{expkv} nor \texttt{exPkv\_cs} keep track of defined keys) – so to make sure there are no conflicts only new definitions are allowed (that’s not the case for individual keys, only for frontend macros).

1.1.1 Primary Keys

In the following descriptions there will be one argument named \langle \texttt{primary keys} \rangle. This argument should be a \langle \texttt{key} \rangle = \langle \texttt{value} \rangle list where each \langle \texttt{key} \rangle will be one primary key and \langle \texttt{value} \rangle the associated initial value. By default all keys are defined short, but you can define long keys by prefixing \langle \texttt{key} \rangle with \texttt{long} (e.g., \texttt{long name=Jonathan P. Spratte}). You only need \texttt{long} if the key should be able to take a \texttt{\par} token. Note however that \texttt{long} keys are a microscopic grain faster (due to some internals of \texttt{exPkv\_cs}). Only if at least one of the keys was \texttt{long} the \langle \texttt{ca} \rangle in the following defining macros will be \texttt{\long}. For obvious reasons there is no possibility to define a macro or key as \texttt{\protected{}).

At the moment \texttt{exPkv\_cs} doesn’t require any internal keys, but I can’t foresee whether this will be the case in the future as well, as it might turn out that some features I deem useful can’t be implemented without such internal keys. Because of this, please don’t use key names starting with \texttt{EKVC1} as that should be the private name space.
1.1.2 Split

The split variants will provide the key values as separate arguments. This limits the
number of keys for which this is truly useful.

\texttt{\textbackslash ekvcSplit}\langle cs\rangle\{\{primary\ keys\}\{definition\}\}

This defines \(cs\) to be a macro taking one mandatory argument which should contain
a \langle key\rangle=\langle value\rangle list. The \langle primary keys\rangle will be defined for this macro (see subsubsection 1.1.1). The \langle definition\rangle is the code that will be executed. You can access the
\langle value\rangle of a \langle key\rangle by using a macro parameter from \#1 to \#9. The order of the macro
parameters will be the order provided in the \langle primary keys\rangle list (so \#1 is the \langle value\rangle
of the key defined first). With \texttt{\textbackslash ekvcSplit} you can define macros using at most nine
primary keys.

\texttt{\textbackslash ekvcSplitAndForward}\langle cs_1\rangle\langle cs_2\rangle\{\{primary\ keys\}\}

This defines \(cs_1\) to be a macro taking one mandatory argument which should contain
a \langle key\rangle=\langle value\rangle list. You can use as many primary keys as you want with this. The
primary keys will be forwarded to \(cs_2\) as braced arguments (as many as necessary for
your primary keys). The order of the braced arguments will be the order of your primary
key definitions.

1.1.3 Hash

The hash variants will provide the key values as a single argument in which you can access
specific values using a special macro. The implementation might be more convenient and
scale better, \textit{but} it is much slower (for a primitive macro with a single key benchmarking
was almost 1.7 times slower, the root of which being the key access with \texttt{\textbackslash ekvcValue},
not the parsing, and for a key access using \texttt{\textbackslash ekvcValueFast} it was still about 1.2 times
slower). So if your macro uses less than ten primary keys, you should most likely use the
split approach.

\texttt{\textbackslash ekvcHash}\langle cs\rangle\{\{primary\ keys\}\{definition\}\}

This defines \(cs\) to be a macro taking one mandatory argument which should contain a
\langle key\rangle=\langle value\rangle list. You can use as many primary keys as you want. The primary keys
will be forwarded as a single argument containing every key to the underlying macro.
The underlying macro is defined as \langle definition\rangle, in which you can access the \langle value\rangle
of a \langle key\rangle by using \texttt{\textbackslash ekvcValue}\langle key\rangle\{\#1\}.

\texttt{\textbackslash ekvcHashAndForward}\langle cs_1\rangle\langle cs_2\rangle\{\{primary\ keys\}\}

This defines \(cs_1\) to be a macro taking one mandatory argument which should contain
a \langle key\rangle=\langle value\rangle list. You can use as many primary keys as you want. The primary
keys will be forwarded as a single argument containing every key to the underlying macro.
For the underlying macro \(cs_2\) is used (so this will provide the key list as a single
argument to \(cs_2\)). In the underlying macro you can access the \langle value\rangle of a \langle key\rangle by
using \texttt{\textbackslash ekvcValue}\langle key\rangle\{\#1\}.
\ekvcValue \(\text{\(\text{\ekvcValue}{\langle key\rangle}{\langle key\ list\rangle}\)}\)
This is a safe (but slow) way to access your keys in a hash variant. \(\langle key\rangle\) is the key which's \(\langle value\rangle\) you want to use out of the \(\langle key\ list\rangle\). \(\langle key\ list\rangle\) should be the key list argument forwarded to your underlying macro by \text{\(\text{\ekvcHash}\)} or \text{\(\text{\ekvcHashAndForward}\)}. It will be tested whether the hash function to access that \(\langle key\rangle\) exists, the \(\langle key\rangle\) argument is not empty, and that the \(\langle key\ list\rangle\) really contains a \(\langle value\rangle\) of that \(\langle key\rangle\). This macro needs exactly two steps of expansion.

\\ekvcValueFast \(\text{\(\text{\ekvcValueFast}{\langle key\rangle}{\langle key\ list\rangle}\)}\)
This behaves just like \text{\(\text{\ekvcValue}\)}, but \textit{without any safety tests}. As a result this is about 1.4 times faster \textit{but} will throw low level \TeX{} errors eventually if the hash function isn’t defined or the \(\langle key\rangle\) isn’t part of the \(\langle key\ list\rangle\) \textit{(e.g., because it was defined as a key for another macro – all macros share the same hash function per \(\langle key\rangle\))}. Use it if you know what you’re doing. This macro needs exactly three steps of expansion in the no-errors case.

1.2 Secondary Keys

To remove some of the limitations with the approach that each primary key matches an argument or hash entry, you can define secondary keys. Those have to be defined for each macro but it doesn’t matter whether that macro was a split or a hash variant. If a secondary key references another key it doesn’t matter whether that other key is primary or secondary.

Secondary keys can have a prefix (like long) which are called \texttt{p-type} prefixes and must have a type (like meta) which are called \texttt{t-type} prefix. Some types might require some \texttt{p-prefixes}, while others might forbid those.

Please keep in mind that key names shouldn’t start with \texttt{EVC|}.

\\ekvcSecondaryKeys \(\text{\(\text{\ekvcSecondaryKeys}{\langle cs\rangle}{\langle key\rangle}=(\langle value\rangle), \ldots\)}\)
This is the front facing macro to define secondary keys. For the macro \(\langle cs\rangle\) define \(\langle key\rangle\) to have definition \(\langle value\rangle\). The general syntax for \(\langle key\rangle\) should be

\(\langle prefix\rangle\ \langle name\rangle\)

Where \(\langle prefix\rangle\) is a space separated list of optional \texttt{p-type} prefixes followed by one \texttt{t-type} prefix. The syntax of \(\langle value\rangle\) is dependent on the used \texttt{t-prefix}.

1.2.1 \texttt{p-type} Prefixes

There is only one \texttt{p-prefix} available, which is long.

long

The following key will be defined \texttt{long}.

1.2.2 \texttt{t-type} Prefixes

If you’re familiar with \texttt{exp\{\texttt{def}} you’ll notice that the \texttt{t-type} prefixes provided here are much fewer. The expansion only concept doesn’t allow for great variety in the auto-defined keys.
The syntax examples of the t-prefixes will show which p-prefix will be automatically used by printing those black \textit{(long)}, which will be available in grey \textit{(long)}, and which will be disallowed in red \textit{(long)}. This will be put flush right next to the syntax line.

\begin{verbatim}
meta ⟨key⟩ = {⟨key⟩=⟨value⟩, ...} long

With a meta key you can set other keys. Whenever ⟨key⟩ is used the keys in the ⟨key⟩=⟨value⟩ list will be set to the values given there. You can use the ⟨value⟩ given to ⟨key⟩ by using \#1 in the ⟨key⟩=⟨value⟩ list. The keys in the ⟨key⟩=⟨value⟩ list can be primary and secondary ones.

nmeta nmeta ⟨key⟩ = {⟨key⟩=⟨value⟩, ...} long

An nmeta key is like a meta key, but it doesn’t take a value, so the ⟨key⟩=⟨value⟩ list is static.

alias alias ⟨key⟩ = ⟨key2⟩ long

This assigns the definition of ⟨key2⟩ to ⟨key⟩. As a result ⟨key⟩ is an alias for ⟨key2⟩ behaving just the same. Both the value taking and the \texttt{NoVal} version (that’s \texttt{expkv} slang for a key not accepting a value) will be copied if they are defined when alias is used. Of course, ⟨key2⟩ has to be defined, be it as a primary or secondary one.

default default ⟨key⟩ = {⟨default⟩} long

If ⟨key⟩ is a defined value taking key, you can define a \texttt{NoVal} version with this that will behave as if ⟨key⟩ was given ⟨default⟩ as its ⟨value⟩. Note that this doesn’t change the initial values of primary keys set at definition time in \texttt{ekvcSplit} and friends. ⟨key⟩ can be a primary or secondary key.

\section{Example}

How could a documentation be a good documentation without some basic examples? Say we want to define a small macro expanding to some character description (who knows why this has to be expandable?). A character description will not have too many items to it, so we use \texttt{ekvcSplit}.

\begin{verbatim}
\ekvcSplit\character
 |
   name=John Doe,
   age=any,
   nationality=the Universe,
   hobby=to exist,
   type=Mister,
   pronoun=He,
   possessive=his,
 |
\%
  #1 is a #5 from #3. #6 is of #2 age and #7 hobby is #4.\par
 |
\end{verbatim}

Also we want to give some short cuts so that it’s easier to describe several persons.
\texttt{\textbackslash ekvcSecondaryKeys\ character}

\begin{verbatim}
\{
alias pro = pronoun, 
alias pos = possessive, 
nmeta me = 
  \{
    name=Jonathan P. Spratte, 
age=a young, 
nationality=Germany, 
hobby=\TeX\ coding, 
  \},
meta lady = 
  \{
type=Lady, pronoun=She, possessive=her, name=Jane Doe, #1, 
\},
nmeta paulo = 
  \{
name=Paulo, 
type=duck, 
age=a young, 
nationality=Brazil, 
hobby=to quack, 
\}
\}
\end{verbatim}

Now we can describe people using
\begin{verbatim}
\character[]
\character{me}
\character{paulo}
\character
  \{lady={name=Evelyn,nationality=Ireland,age=the best,hobby=reading}}
\character
  \{
    name=Our sun, type=star, nationality=our solar system, pro=It, 
age=an old, pos=its, hobby=shining 
  \}
\end{verbatim}

As one might see, the lady key could actually have been an nmeta key as well, as all that
is done with the argument is using it as a \texttt{⟨key⟩=⟨value⟩} list.

Utilizing \texttt{\textbackslash xparse} and forwarding arguments one can easily define \texttt{⟨key⟩=⟨value⟩}
macros with actual optional and mandatory arguments as well. A small nonsense exam-
ple (which should perhaps use \texttt{\textbackslash ekvcSplitAndForward} instead of \texttt{\textbackslash ekvcHashAndForward}
since it only uses four keys and one other argument – and isn't expandable since it uses a 
tabular environment):
\begin{verbatim}
\usepackage{\textbackslash xparse}
\makeatletter
\NewExpandableDocumentCommand\nonsense[O{ } m]{\nonsense@a[#1][#2]}
\end{verbatim}

\begin{verbatim}
\texttt{\textbackslash ekvcHashAndForward\nonsense@a\nonsense@b}
\{
keyA = A, 
keyB = B, 
keyC = c, 
\}
\end{verbatim}

6
\newcommand\nonsense@b[2]{%
\begin{tabular} {llll}
key & A & \ekvcValue{keyA} & #1 \\
& B & \ekvcValue{keyB} & #1 \\
& C & \ekvcValue{keyC} & #1 \\
& D & \ekvcValue{keyD} & #1 \\
\multicolumn{2}{l}{\text{mandatory}} & #2 \\
\end{tabular}\par
}\makeatother

And then we would be able to do some nonsense:
\nonsense{ }
nonsense{keyA=hihi }{haha}
nonsense{keyA=hihi, keyB=A}{hehe}
nonsense{keyC=huhu, keyA=hihi, keyB=A}{haha}

1.4 Useless Macros

Perhaps these macros aren’t completely useless, but I figured from a user’s point of view I wouldn’t know what I should do with these.

\ekvcDate \ekvcVersion

These two macros store the version and the date of the package/generic code.

1.5 Bugs

Of course I don’t think there are any bugs (who would knowingly distribute buggy software as long as he isn’t a multi-million dollar corporation?). But if you find some please let me know. For this one might find my email address on the first page or file an issue on Github: https://github.com/Skillmon/tex_expkv-cs

1.6 License

Copyright © 2020 Jonathan P. Spratte

This work may be distributed and/or modified under the conditions of the \LaTeX{} Project Public License (LPPL), either version 1.3c of this license or (at your option) any later version. The latest version of this license is in the file: http://www.latex-project.org/lppl.txt

This work is “maintained” (as per LPPL maintenance status) by Jonathan P. Spratte.
2 Implementation

2.1 The \LaTeX Package

Just like for \texttt{expkv} we provide a small \LaTeX package that sets up things such that we behave nicely on \LaTeX packages and files system. It’ll \texttt{\input} the generic code which implements the functionality.

\begin{verbatim}
\RequirePackage{expkv}
\def\ekvc@tmp
{%
 \ProvidesFile{expkv-cs.tex}%
  [%
   \ekvcDate\space v\ekvcVersion\space
   define expandable key=val macros using expkv%
  %]
%
\input{expkv-cs.tex}
\ProvidesPackage{expkv-cs}%
  [%
   \ekvcDate\space v\ekvcVersion\space
   define expandable key=val macros using expkv%
  ]
\end{verbatim}

2.2 The Generic Code

The rest of this implementation will be the generic code.

Load \texttt{expkv} if the package didn’t already do so – since \texttt{expkv} has safeguards against being loaded twice this does no harm and the overhead isn’t that big. Also we reuse some of the internals of \texttt{expkv} to save us from retyping them.

\begin{verbatim}
\input expkv
We make sure that \texttt{expkv-cs.tex} is only input once:
\expandafter\ifx\csname ekvcVersion\endcsname\relax
  \else
  \expandafter\endinput
\fi
\end{verbatim}

\texttt{\ekvcVersion} and \texttt{\ekvcDate} are only defined on our first input, so let’s store the version and date in a macro.

\begin{verbatim}
\def\ekvcVersion{0.1}
\def\ekvcDate{2020-04-04}
\end{verbatim}

(End definition for \texttt{\ekvcVersion} and \texttt{\ekvcDate}. These functions are documented on page 7.)

If the \LaTeX format is loaded we want to be a good file and report back who we are, for this the package will have defined \texttt{\ekvc@tmp} to use \texttt{\ProvidesFile}, else this will expand to a \texttt{\relax} and do no harm.

\begin{verbatim}
\csname ekvc@tmp\endcsname
Store the category code of \texttt{@} to later be able to reset it and change it to 11 for now.
\end{verbatim}

\begin{verbatim}
\expandafter\if\csname ekvc@tmp\endcsname=\catcode'\@\relax
  \catcode'\@=11
\fi
\end{verbatim}

\texttt{\ekvc@tmp} will be reused later, but we don’t need it to ever store information long-term after \texttt{expkv\textunderscore cs} was initialized.
\ekvc@keycount
We’ll need to keep count how many keys must be defined for each macro in the split
variants.
26 \newcount\ekvc@keycount
(End definition for \ekvc@keycount.)
\ekvc@long
Some macros will have to be defined long. These two will be let to \long when this
should be the case.
27 \def\ekvc@long{}
28 \def\ekvc@any@long{}
(End definition for \ekvc@long and \ekvc@any@long.)
\ekvcSplitAndForward
The first user macro we want to set up can be reused for \ekvcSplit. We’ll split this one
up so that the test whether the macro is already defined doesn’t run twice.
29 \protected\long\def\ekvcSplitAndForward#1#2#3{%
30 \ekv@ifdefined{\expandafter\@gobble\string#1}{%
31 \ekvc@err@already@defined{#1}%
32 \{\ekvcSplitAndForward@{#1}{#2}{#3}%
33 }
34 }
(End definition for \ekvcSplitAndForward. This function is documented on page 3.)
\ekvcSplitAndForward@0
The actual macro setting up things. We need to set some variables, forward the key
list to \ekvc@SetupSplitKeys, and afterwards define the front facing macro to call
\ekvset and put the initials and the argument sorting macro behind it. The internals \ekvc@any@long, \ekvc@initials and \ekvc@keycount will be set correctly by
\ekvc@SetupSplitKeys.
35 \protected\long\def\ekvcSplitAndForward@0#1#2#3{%
36 \{%
37 \edef\ekvc@set{\string#1}%
38 \ekvc@SetupSplitKeys{#3}%
39 \ekvc@any@long\edef\#1#1%
40 \%\unexpanded{\ekvset}{\ekvc@set}{#1}%
41 \unexpanded{\expandafter}
42 \{\csname ekvc@split\the\ekvc@keycount\endcsname}%
43 \unexpanded{\expandafter}{\ekvc@initials{}#2}%
44 }
45 }
(End definition for \ekvcSplitAndForward@0.)
\ekvcSplit
The first half is just \ekvcSplitAndForward then we define the macro to which the parsed key list is forwarded. There we need to allow for up to nine arguments.
47 \protected\long\def\ekvcSplit#1#2#3{%
48 \{%
49 \ekv@ifdefined{\expandafter\@gobble\string#1}{%
50 \{\ekvc@err@already@defined{#1}%
51 \%
52 \expandafter
53 \ekvcSplitAndForward@0\csname ekvc\string#1\endcsname\#2}%
54 }\ifnum\ekvc@keycount=0
These macros parse the list of keys and set up the key macros. First we need to initialise some macros and start \ekvparses.

\protected\long\def\ekvc@SetupSplitKeys#1{% 
\% \ekvc@keycount=0 
\def\ekvc@any@long{}% 
\def\ekvc@initials{}% 
\ekvparses\ekvc@err@value@required\ekvc@SetupSplitKeys@a{#1}%
}%

Then we need to step the key counter for each key. Also we have to check whether this key has a long prefix so we initialise \ekvc@long.

\protected\def\ekvc@SetupSplitKeys@a#1{% 
\% \advance\ekvc@keycount1 
\def\ekvc@long{}% 
\ekvc@ifspace{#1}{\ekvc@SetupSplitKeys@b#1\ekvc@stop}{\ekvc@SetupSplitKeys@c{#1}}%
}%

If there was a space, there might be a prefix. If so call the prefix macro, else call the next step \ekvc@SetupSplitKeys@c which will define the key macro and add the key's value to the initials list.

\protected\def\ekvc@SetupSplitKeys@c#1#2{% 
\% \ekvc@long\def\ekvc@SetupSplitKeys@c@1#2\% 
\% \ekvc@long\def\ekvc@SetupSplitKeys@c@2#2\% 
}%

The inner definition is grouped, because we don't want to actually define the marks we build with \csname. We have to append the value to the \ekvc@initials list here with the correct split mark. The key macro will read everything up to those split marks and change the value following it to the value given to the key. Additionally we'll need a sorting macro for each key count in use so we set it up with \ekvc@Setup@splitmacro.
\begin{verbatim}
\begingroup
\edef\ekvc@tmp
\endgroup
\long\def\unexpanded{\ekvc@tmp}
\unexpanded\expandafter
\csname ekvc@splitmark@\the\ekvc@keycount\endcsname
\unexpanded\expandafter
\csname ekvc@splitmark@\the\ekvc@keycount\endcsname{#2}
\endverbatim

The short variant needs a bit of special treatment. The key macro will be short to throw the correct error, but since there might be long macros somewhere the reordering of arguments needs to be long, so for short keys we use a two step approach, first grabbing only the short argument, then reordering.

\begin{verbatim}
\unless\ifx\ekvc@long\long
\let\unexpanded\expandafter
\csname ekvc@set(#1)\endcsname\ekvc@tmp
\def\unexpanded{\ekvc@tmp}{#1}
\fi
\def\unexpanded{\ekvc@initials}
\unexpanded\expandafter\csname ekvc@splitmark@\the\ekvc@keycount\endcsname{#2}
\endverbatim

(End definition for \ekvc@SetupSplitKeys and others.)

\ekvc@split@p@long The long prefix lets the internals \ekvc@long and \ekvc@any@long to \long so that the key macro will be long.

\begin{verbatim}
\protected\def\ekvc@split@p@long
\let\ekvc@long\long
\let\ekvc@any@long\long
\ekvc@SetupSplitKeys@c
\endverbatim

(End definition for \ekvc@split@p@long.)

\ekvc@defarggobbler This is needed to define a macro with 1-9 parameters programmatically. \LaTeX's \newcommand does something similar for example.

\begin{verbatim}
\protected\def\ekvc@defarggobbler#1{\def\ekvc@tmp##1#1##{##1#1}}
\end{verbatim}
Since the first split macro is different from the others, we manually set that one up now. All the others will be defined as needed (always globally). The split macros just read up until the correct split mark, move that argument into a list and reinsert the rest, calling the next split macro afterwards.

\begin{verbatim}
\edef\ekvc@tmp\{\long\gdef\unexpanded\expandafter{\csname ekvc@split@1\endcsname}\%
\unexpanded\expandafter{\csname ekvc@splitmark@1\endcsname}\%
##1##2##3%
##3{##1}##2%
\}
\ekvc@tmp
\end{verbatim}

\protected\def\ekvc@setup@splitmacro#1\{\ekv@ifdefined{ekvc@split@#1}{}{\begingroup\edef\ekvc@tmp\{\long\gdef\expandafter{\csname ekvc@split@#1\endcsname}####1%
\unexpanded\expandafter{\csname ekvc@splitmark@#1\endcsname}####2####3%
{\expandafter{\csname ekvc@split@\numexpr#1-1\relax\endcsname}####1{{####2}####3}}\}\ekvc@tmp\endgroup}\}

\end{verbatim}

(End definition for \ekvc@setup@splitmacro and \ekvc@split@1.)

\ekvcHashAndForward

\ekvcHashAndForward works just like \ekvcSplitAndForward.

\begin{verbatim}
\protected\long\def\ekvcHashAndForward#1#2#3\{\ekv@ifdefined{\expandafter\@gobble\string#1}\{\ekvc@err@already@defined{#1}\}
{\expandafter{\expandafter\ekvcHashAndForward@{#1}{#2}{#3}}}
\end{verbatim}

(End definition for \ekvcHashAndForward. This function is documented on page 3.)

\ekvcHashAndForward@0

This is more or less the same as \ekvcHashAndForward@0. Instead of an empty group we place a marker after the initials, we don’t use the sorting macros of split, but instead pack all the values in one argument.
\protected\long\def\ekvcHashAndForward@#1#2#3\{% 
  \edef\ekvc@set{\string#1}\%  
  \ekvc@SetupHashKeys(#3)\%  
  \ekvc@any@long\def#1##1\{% 
    \unexpanded{\ekvset}{\ekvc@set}{##1}\%  
    \unexpanded{\ekvc@hash@pack@argument}\%  
    \unexpanded{\expandafter{\ekvc@initials\ekvc@stop#2}}\%  
  \}  
\}  
\strip\long\def\ekvcHash#1#2#3\{%}  
\strip\long\def\ekvcHashAndForward@{\string#1}\%  
\strip\long\def\ekvcSplit#1#2{\string#1\%}  
\strip\long\def\ekvc@SetupHashKeys#1\{%  
  \def\ekvc@any@long\{}  
  \def\ekvc@initials\{}  
  \ekvparse{\ekvcerr@value@required}{\ekvc@SetupHashKeys@a\string#1}\%  
\}  
\strip\long\def\ekvc@SetupHashKeys@a#1\{%  
  \def\ekvc@long\{}  
  \ekvc@ifspace{#1}{\ekvc@SetupHashKeys@b#1\ekvc@stop}{\ekvc@SetupHashKeys@c{#1}}\%  
\}  
\strip\long\def\ekvc@SetupHashKeys@b#1 #2\ekvc@stop\{%  
  \ekv@ifdefined{ekvc@hash@p@#1}{\csname ekvc@hash@p@#1\endcsname{#2}}{\ekvc@SetupHashKeys@c{#1 #2}}\%  
\}  
\strip\long\def\ekvc@hash@pack@argument#1\ekvc@stop#2{#2{#1}}  
\strip\long\def\ekvc@hash@pack@argument\%  
\strip\long\def\ekvc@SetupHashKeys@0a\{%  
\strip\long\def\ekvc@SetupHashKeys@0b\{%  
% The function \ekvcHash does the same as \ekvcSplit, but has the advantage of not needing to count arguments, so the definition of the internal macro is a bit more straight forward.  
\strip\long\def\ekvcHash@\%  
% All this macro does is pack the values into one argument and forward that to the next macro.  
\strip\long\def\ekvc@hash@pack@argument@\%  
% This should look awfully familiar as well, since it's just the same as for the split keys with a few other names here and there.  
\strip\long\def\ekvc@SetupHashKeys@0\%
Yes, even the defining macro looks awfully familiar. Instead of numbered we have named marks. Still the key macros grab everything up to their respective mark and reorder the arguments. The same quirk is applied for short keys. And instead of the \ekvc@setup@splitmacro we use \ekvc@setup@hashmacro.

\begin{verbatim}
\protected\long\def\ekvc@SetupHashKeys@c#1#2\%
  {%
  \begingroup
  \edef\ekvc@tmp{%
    \endgroup
    \long\def\unexpanded{%
      \expandafter{\csname ekvc@hashmark@#1\endcsname}####1####2%
      \unexpanded\expandafter{\csname ekvc@hashmark@#1\endcsname}####3%
    }%
    \unless\ifx\ekvc@long\long
      \let\unexpanded\expandafter{\csname ekvc@set(#1)\endcsname}\ekvc@tmp
      \def\unexpanded{%
        \expandafter{\csname ekvc@set(#1)\endcsname}{####1}%
        {####2}%
        \unexpanded\expandafter{\csname ekvc@hashmark@#1\endcsname}{####1}%
      }%
      \fi
    \def\unexpanded{%
      \expandafter{\csname ekvc@hashmark@#1\endcsname}#2%
      {####1}%
    }%
  }% \ekvc@setup@hashmacro{#1}%
  \ekvc@tmp
  \ekvlet\ekvc@set{#1}\ekvc@tmp
  }
\end{verbatim}

(End definition for \ekvc@SetupHashKeys, \ekvc@SetupHashKeys@a, and \ekvc@SetupHashKeys@b.)

\ekvc@setup@hashmacro Nothing astonishing here either.

\begin{verbatim}
\protected\def\ekvc@hash@p@long
  {%
    \let\ekvc@long\long
    \let\ekvc@any@long\long
    \ekvc@setup@hashmacro{#1}%
  }
\end{verbatim}

(End definition for \ekvc@hash@p@long.)

\ekvc@setup@hashmacro The safe hash macros will be executed inside of a \romannumeral expansion context, so they have to insert a stop mark for that once they are done. Most of the tests which have to be executed will already be done, but we have to play safe if the hash doesn’t show up in the hash list. Therefore we use some \ekvc@marks and \ekvc@stop to throw errors if the hash isn’t found in the right place. The fast variants have an easier life and just return the correct value.
\long\def\ekvcValue#1#2\%  
{\romannumeral'\^^@\%  
  \ekv@ifdefined{ekvc@safehash@#1}\{%  
  \csname ekvc@safehash@#1\endcsname{#2}  
  \ekvc@err@unknown@hash{#1}  
  \}  
}\%  
\long\def\ekvcValue#1#2\%  
{\romannumeral'\^^@\%  
  \ekv@ifdefined{ekvc@safehash@#1}\{%  
  \csname ekvc@safehash@#1\endcsname{#2}  
  \ekvc@err@unknown@hash{#1}  
  \}  
}\%  
\endgroup  
\}  
(End definition for \ekvcValue. This function is documented on page 4.)

\ekvcValue  
All this does is a few consistency checks on the first argument (not empty, hash macro exists) and then call that hash-grabbing macro that will also test whether the hash is inside of #2 or not.

(End definition for \ekvc@setup@hashmacro.)
\textbf{\texttt{\textbackslash ekvcValueFast}} To be as fast as possible, this doesn’t test for anything, assuming the user knows best.

\begin{verbatim}
\long\def\ekvcValueFast#1#2{\csname ekvc@fasthash@#1\endcsname#2}\ekvc@stop
\end{verbatim}

(\textit{End definition for \texttt{\textbackslash ekvcValueFast}. This function is documented on page 4.)

\textbf{\texttt{\textbackslash ekvc\texttt{safehash@}} \texttt{\textbackslash ekvc\texttt{fasthash@}}}

At least in the empty hash case we can provide a meaningful error message without affecting performance by just defining the macro that would be build in that case. There is of course a downside to this, the error will not be thrown by \texttt{\textbackslash ekvcValueFast} in three expansion steps. The safe hash variant has to also stop the roman numeral expansion.

\begin{verbatim}
\long\def\ekvc@safehash@#1{\ekvc@err@empty@hash\@gobble{}}% keep this space
\long\def\ekvc@fasthash@#1\ekvc@stop{\ekvc@err@empty@hash}
\end{verbatim}

(\textit{End definition for \texttt{\textbackslash ekvc\texttt{safehash@}} and \texttt{\textbackslash ekvc\texttt{fasthash@}}.)

\textbf{\texttt{\textbackslash ekvcSecondaryKeys}}

The secondary keys are defined pretty similar to the way the originals are, but here we also introduce some key types (those have a @t@ in their name) additionally to the prefixes.

\begin{verbatim}
\protected\long\def\ekvcSecondaryKeys#1#2{%
  \edef\ekvc@set{\string#1}%
  \ekvparse\ekvc@err@value@required\ekvcSecondaryKeys@a{#2}%
}%
\protected\def\ekvcSecondaryKeys@a#1{%
  \def\ekvc@long{}%
  \ekvc@ifspace{#1}%
  {\ekvcSecondaryKeys@b#1\ekvc@stop}%
  {\ekvc@err@missing@type{#1}\@gobble}%
}%
\protected\def\ekvcSecondaryKeys@b#1 #2\ekvc@stop{%
  \ekv@ifdefined{ekvc@p@#1}{\csname ekvc@p@#1\endcsname}\%
  \ekv@ifdefined{ekvc@t@#1}{\csname ekvc@t@#1\endcsname}\%
  {\ekvc@err@unknown@keytype{#1}\#2}\%
}%
\end{verbatim}

(\textit{End definition for \texttt{\textbackslash ekvcSecondaryKeys}. This function is documented on page 4.)

2.2.1 \textbf{Secondary Key Types}

\textbf{\texttt{\textbackslash ekvc\texttt{p@long}} \texttt{\textbackslash ekvc\texttt{after@ptype}}}

The prefixes are pretty straight forward again. Just set \texttt{\textbackslash ekvc\texttt{p@long}} and forward to the @t@ type.

\begin{verbatim}
\protected\def\ekvc@p@long#1{%
  \ekvc@after@ptype#1\ekvc@stop}
\end{verbatim}
The \texttt{meta} and \texttt{nmeta} key types use a nested \texttt{ekvset} to set other keys in the same macro's \texttt{⟨set⟩}.

\begin{verbatim}
\protected\def\ekvc@t@meta
\edef\ekvc@tmp{\ekvc@set}
\expandafter\ekvc@type@meta\expandafter{\ekvc@tmp}\ekvc@long{##1}\ekvlet
\protected\def\ekvc@t@nmeta#1%
\edef\ekvc@tmp{\ekvc@set}
\expandafter\ekvc@type@meta\expandafter{\ekvc@tmp}{}\ekvletNoVal{#1}%
\protected\long\def\ekvc@type@meta#1#2#3#4#5#6%
#2\def\ekvc@tmp#3{\ekvset{#1}{#6}}
#4\ekvc@set{#5}\ekvc@tmp
\end{verbatim}

\textit{(End definition for \texttt{ekvc@t@meta}, \texttt{ekvc@t@nmeta}, and \texttt{ekvc@type@meta}.)}

\texttt{alias} just checks whether there is a key and/or \texttt{NoVal} key defined with the target name and \texttt{let} the key to those.

\begin{verbatim}
\protected\def\ekvc@t@alias#1#2%
\edef\ekvc@tmp{\ekvc@set}
\expandafter\ekvc@type@meta\expandafter{\ekvc@tmp}{}\ekvletNoVal{#1}%
#2\def\ekvc@tmp#3{\ekvset{#1}{#6}}
#4\ekvc@set{#5}\ekvc@tmp
\end{verbatim}

\textit{(End definition for \texttt{ekvc@t@alias}.)}
The \texttt{default} key can be used to set a \texttt{NoVal} key for an existing key. It will just pass the \texttt{⟨value⟩} to the key macro of that other key.

\begin{verbatim}
\protected\long\def\ekvc@t@default#1#2\%
{\ekvifdefined\ekvc@set{#1}\%
{\ekvc@assert@not@long{default #1}\%
\edef\ekvc@tmp{\unexpanded\expandafter{%\csname\ekv@name\ekvc@set{#1}\endcsname{#2}}}%
\ekvletNoVal\ekvc@set{#1}\ekvc@tmp
}\%
{\ekvc@err@unknown@key{#1}}%
}
\end{verbatim}

(End definition for \texttt{\ekvc@t@default}.)

\subsection*{Helper Macros}

\begin{verbatim}
\long\def\ekvc@ifspace#1\%
{\ekvc@ifspace@#1 \ekv@ifempty@B \ekv@ifempty@false\ekv@ifempty@A\ekv@ifempty@B\@firstoftwo}
\long\def\ekvc@ifspace@#1 \% keep this space
{\ekv@ifempty@\ekv@ifempty@A}
\end{verbatim}

(End definition for \texttt{\ekvc@ifspace} and \texttt{\ekvc@ifspace@}.)

\subsection*{Assertions}

\begin{verbatim}
\long\def\ekvc@assert@not@long#1\%
{\ifx\ekvc@long\long\ekvc@err@no@long{#1}\fi}
\end{verbatim}

(End definition for \texttt{\ekvc@assert@not@long}.)

\subsection*{Messages}

\begin{verbatim}
\protected\def\ekvc@err@toomany\#1\%
{\errmessage{expkv-cs Error: Too many keys for macro \texttt{⟨string#1⟩}}}
\protected\def\ekvc@err@value@required\#1\%
{\errmessage{expkv-cs Error: Missing value for key \texttt{⟨unexpanded{#1}⟩}}}
\protected\def\ekvc@err@missing@type\#1\%
\end{verbatim}

Boring unexpandable error messages.
We need a way to throw error messages expandably in some contexts.

```
\begingroup
\edef\ekvc@err{
\endgroup
\unexpanded{\long\def\ekvc@err{% #1\%
\unexpanded{\expandafter\ekvc@err\@firstofone}%
\unexpanded{\expandafter\csname ! expkv-cs Error: \endcsname}#1.}%
\unexpanded{\ekv@stop}%%
}%
\ekvc@err
\def\ekvc@err@{
\expandafter\ekv@gobbleto@stop}
```

(End definition for `\ekvc@err@toomany and others.)

```
\ekvc@err
\ekvc@err@
\begin{group}
\edef\ekvc@err
{\endgroup
\unexpanded{\long\def\ekvc@err##1{% #1\%
\unexpanded{\expandafter\ekvc@err\@firstofone}%
\unexpanded{\expandafter\csname ! expkv-cs Error: \endcsname}#1.}%
\unexpanded{\ekv@stop}%%
}%
\ekvc@err
\def\ekvc@err@{\expandafter\ekv@gobbleto@stop}
```

(End definition for `\ekvc@err@unknown@hash, `\ekvc@err@empty@hash, `\ekvc@err@missing@hash, `\ekvc@err@unknown@hash, `\ekvc@err@empty@hash, and `\ekvc@err@missing@hash.)

Now everything that’s left is to reset the category code of \@.

```
\catcode`\@=\ekvc@tmp
```
The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

A

alias ........................................ 5

D

default ........................................ 5

E

\ekvcDate ..................................... 6, 7, 13, 21
\ekvcHash ............................. 3, 181
\ekvcHashAndForward .............. 3, 164
\ekvcSecondaryKeys ..................... 4, 298
\ekvcSplit ........................... 3, 347
\ekvcSplitAndForward .......... 3, 29
\ekvcValue ................................ 4, 288
\ekvcValueFast ..................... 4, 295
\ekvcVersion .................... 6, 7, 13, 21
\ekvifdefined ...................... 356, 372
\ekvifdefinedNoVal ................. 362
\ekvlet ................................ 122, 239, 339
\ekvletkv ................................ 358
\ekvletkNoVal .............. 345, 380
\ekvparse .......... 75, 196, 301
\ekvset .......... 41, 176, 349

L

long ........................................ 4

M

meta ........................................ 5

N

nmeta ........................................ 5

T

T\TeX\ and \LaTeX\ \texttt{2e} commands:
\ekv@gobbleto@stop .............. 437
\ekv@ifdefined ............. 31, 49, 87, 144
166, 183, 207, 250, 291, 312, 315, 332
\ekv@isempty................ 391
\ekv@isempty@A ...... 387, 391
\ekv@isempty@B .......... 386, 387
\ekv@isempty@false .... 387
\ekv@name ......................... 378
\ekv@stop .......................... 433
\ekv@after@ptype ........... 321
\ekv@any@long ........... 27, 39, 64, 73, 128, 174, 188, 194, 245
\ekv@assert@not@long ............ 343, 354, 374, 393
\ekv@defarggobbler ........... 59, 61, 131
\ekv@err ........................... 425, 438, 439, 440
\ekv@err@ ................ 425
\ekv@err@already@defined .......... 32, 50, 167, 184, 394
\ekv@err@empty@hash ....... 296, 297, 338
\ekv@err@missing@hash ...... 270, 438
\ekv@err@missing@type ... 308, 328, 394
\ekv@err@no@long ........... 393, 407
\ekv@err@toomany .......... 58, 394
\ekv@err@unknown@hash ...... 293, 438
\ekv@err@unknown@key .... 368, 382, 420
\ekv@err@unknown@key@type .... 317, 334, 416
\ekv@err@value@required .... 75, 196, 301, 394
\ekv@fasthash@ ........... 296
\ekv@hash@p@long ........... 242
\ekv@hash@pack@argument ... 177, 191
\ekv@ifspace .......... 81, 201, 306, 323, 384
\ekv@ifspace@ ........... 384
\ekv@initials ........................ 44, 74, 114, 116, 178, 195, 232, 234
\ekv@keycount ................... 26, 43, 54, 57, 61, 72, 79, 99, 103, 118, 123
\ekv@long ........ 27, 80, 105, 127, 200, 223, 244, 305, 325, 339, 393
\ekv@mark ........ 266, 271, 278
\ekv@p@long ........... 321
\ekv@safehash@ .......... 296
\ekv@set ................................ 240, 248
\ekv@setup@hashmacro .... 123, 132
\ekv@setup@splitmacro ........ 173, 192
\ekv@Setup@HashKeys .... 192
\ekv@Setup@HashKeys@A .......... 192
\ekv@Setup@HashKeys@B .. 203, 209, 211, 246
\ekv@Setup@SplitKeys ...... 38, 70