define expandable \langle key \rangle=\langle value \rangle macros using \texttt{expkv}

Jonathan P. Spratte\textsuperscript{*}

\texttt{2020-04-05 v0.2}

Abstract

\texttt{expkvcs} provides two small interfaces to define expandable \langle key \rangle=\langle value \rangle macros using \texttt{expkv}. It therefore lowers the entrance boundary to expandable \langle key \rangle=\langle value \rangle macros. The stylised name is \texttt{expkvcs} but the files use \texttt{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 ................................ 5
    1.2.2 t-type Prefixes ................................ 5
  1.3 Example ............................................ 6
  1.4 Speed Considerations ................................ 7
  1.5 Useless Macros ................................... 9
  1.6 Bugs ............................................ 9
  1.7 License .......................................... 9

2 Implementation ........................................ 10
  2.1 The \LaTeX\ Package ................................. 10
  2.2 The Generic Code .................................. 10
    2.2.1 Secondary Key Types .......................... 20
    2.2.2 Helper Macros ................................. 21
    2.2.3 Assertions ................................... 22
    2.2.4 Messages .................................... 22

Index ..................................................... 24

\textsuperscript{*}jspratte@yahoo.de
1 Documentation

The \texttt{expkv} package enables the new possibility of creating \langle key\rangle=\langle value\rangle macros which are fully expandable. The creation of such macros is however cumbersome for the average user. \texttt{expkvcs} tries to step in here. It provides interfaces to define \langle key\rangle=\langle value\rangle macros without worrying too much about the implementation. In case you’re wondering now, the ca in \texttt{expkvcs} stands for control sequence, because \texttt{def} was already taken by \texttt{expkvdef} and “control sequence” is the term D. E. Knuth used in his \textsc{TeX}book for named commands hence macros (though he also used the term “macro”). So \texttt{expkvcs} defines control sequences for and with \texttt{expkv}.

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 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{expkvcs}. The interface has some overhead (though it still can be considered fast – check \textsection{1.3}) and the approach has its limits in versatility. If you don’t need to be expandable, you should consider either defining your keys manually using \texttt{expkv} or using \texttt{expkvdef} for convenience. Or you resort to another \langle key\rangle=\langle value\rangle interface.

\texttt{expkvcs} 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{expkvcs} have to be previously undefined or have the \verb|\meaning| of \verb|\relax|. This is necessary as there is no way to undefine keys once they are set up (neither \texttt{expkv} nor \texttt{expkvcs} 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 primary\ keys\rangle. This argument should be a \langle key\rangle=\langle value\rangle list where each \langle key\rangle will be one primary key and \langle value\rangle the associated initial value. By default all keys are defined short, but you can define long keys by prefixing \langle 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{expkvcs}). Only if at least one of the keys was \texttt{long} the \langle 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 eKvC 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 EKVC 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.

\[
\text{\texttt{ekvcSplit}}(\langle cs \rangle\{\langle primary\ keys \rangle\}\{\langle definition \rangle\})
\]

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{ekvcSplit} you can define macros using at most nine primary keys.

\[
\text{\texttt{ekvcSplitAndForward}}(\langle cs_1 \rangle)(\langle cs_2 \rangle)\{\langle primary\ keys \rangle\})
\]

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, 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{ekvcValue}, not the parsing, and for a key access using \texttt{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.

\[
\text{\texttt{ekvcHash}}(\langle cs \rangle)\{\langle primary\ keys \rangle\}\{\langle definition \rangle\})
\]

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{ekvcValue}(\langle key \rangle){#1}. 

3
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 \( \texttt{ekvcValue} \{ \langle key \rangle \} \{#1\} \).

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 \( \texttt{ekvcHash} \) or \( \texttt{ekvcHashAndForward} \). 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.

This behaves just like \( \texttt{ekvcValue} \), but without any safety tests. As a result this is about 1.4 times faster 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 \( \langle key list \rangle \) (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.

If you need a specific \( \langle key \rangle \) from a \( \langle key list \rangle \) more than once, it’ll be a good idea to only extract it once and from then on keep it as a separate argument. Hence the macro \( \texttt{ekvcValueSplit} \) will extract one specific \( \langle key \rangle \)’s value from the list and forward the remainder of the list as the first and the \( \langle key \rangle \)’s value as the second argument to \( \langle next \rangle \), so the result of this will be \( \langle next \rangle \{ \langle key list' \rangle \} \{ \langle value \rangle \} \) with \( \langle key list' \rangle \) the remaining list. This is almost as fast as \( \texttt{ekvcValue} \) and runs the same tests. Keep in mind that you can’t fetch for the same \( \langle key \rangle \) again from \( \langle key list' \rangle \) as it got removed.

This behaves just like \( \texttt{ekvcValueSplit} \), but it won’t run the same tests, hence it is faster but more error prone, just like the relation between \( \texttt{ekvcValue} \) and \( \texttt{ekvcValueFast} \).

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 p-type prefix and must have a type (like `meta`) which are called t-type prefix. Some types might require some p-prefixes, while others might forbid those.
Please keep in mind that key names shouldn’t start with EKVCl.

\texttt{\textbackslash ekvcSecondaryKeys} \texttt{\textbackslash ekvcSecondaryKeys\{cs\}\{key\}=\{value\}, ...}

This is the front facing macro to define secondary keys. For the macro \texttt{(cs)} define \texttt{(key)} to have definition \texttt{(value)}. The general syntax for \texttt{(key)} should be

\texttt{(prefix)\{name\}}

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

1.2.1 p-type Prefixes

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

\texttt{\textbackslash long}

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

1.2.2 t-type Prefixes

If you’re familiar with exPkv\texttt{def} you’ll notice that the 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 (\texttt{long}), which will be available in grey (\texttt{long}), and which will be disallowed in red (\texttt{long}). This will be put flush right next to the syntax line.

\texttt{\textbackslash meta\{key\} = \{(key)=\{value\}, ...\}}

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

\texttt{\textbackslash nmeta\{key\} = \{(key)=\{value\}, ...\}}

An nmeta key is like a meta key, but it doesn’t take a value, so the \texttt{(key)=\{value\}} list is static.

\texttt{\textbackslash alias\{key\} = \{key\}_2}

This assigns the definition of \texttt{(key\_2)} to \texttt{(key)}. As a result \texttt{(key)} is an alias for \texttt{(key\_2)} behaving just the same. Both the value taking and the NoVal version (that’s exPkv\texttt{def} slang for a key not accepting a value) will be copied if they are defined when \texttt{alias} is used. Of course, \texttt{(key\_2)} has to be defined, be it as a primary or secondary one.

\texttt{\textbackslash default\{key\} = \{(default)\}}

If \texttt{(key)} is a defined value taking key, you can define a NoVal version with this that will behave as if \texttt{(key)} was given \texttt{(default)} as its \texttt{(value)}. Note that this doesn’t change the initial values of primary keys set at definition time in \texttt{\textbackslash ekvcSplit} and friends. \texttt{(key)} can be a primary or secondary key.
1.3 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}
\texttt{ekvcSplit} \texttt{character}
  \{
    name=John Doe,
    age=any,
    nationality=the Universe,
    hobby=to exist,
    type=Mister,
    pronoun=He,
    possessive=his,
  \}
\%
\texttt{\texttt{#1} is a \texttt{#5} from \texttt{#3}. \texttt{#6} is of \texttt{#2} age and \texttt{#7} hobby \texttt{is} \texttt{\#4}.\par}
\}
\end{verbatim}

Also we want to give some short cuts so that it's easier to describe several persons.

\begin{verbatim}
\texttt{ekvcSecondaryKeys} \texttt{character}
  \{
    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}
\texttt{character{}}
\texttt{character{me}}
\texttt{character{paulo}}
\texttt{character}
  \{
    lady={name=Evelyn, nationality=Ireland, age=the best, hobby=reading}}
\end{verbatim}
\character

{name=Our sun, type=star, nationality=our solar system, pro=It, age=an old, pos=its, hobby=shining}

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

Using sparse and forwarding arguments one can easily define \langle \texttt{key} \rangle=\langle \texttt{value} \rangle macros with actual optional and mandatory arguments as well. A small nonsense example (which should perhaps use \texttt{ekvcSplitAndForward} instead of \texttt{ekvcHashAndForward} since it only uses four keys and one other argument – and isn't expandable since it uses a tabular environment):

\begin{Verbatim}
usepackage{xparse}
\makeatletter
NewExpandableDocumentCommand\nonsense[O ] m] \nonsense@a[\#1][\#2]
\ekvcHashAndForward nonsense@a
\nonsense@b
\makeatletter
\newcommand\nonsense@b[2]
\%
\begin{tabular}{ l l l}
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
\end{Verbatim}

And then we would be able to do some nonsense:

\begin{Verbatim}
\nonsense[]
nonsense[\texttt{keyA}=\texttt{hihi}][\texttt{haha}]
nonsense[\texttt{keyA}=\texttt{hihi}, \texttt{keyB}=\texttt{A}][\texttt{hehe}]
nonsense[\texttt{keyC}=\texttt{huhu}, \texttt{keyA}=\texttt{hihi}, \texttt{keyB}=\texttt{A}][\texttt{haha}]
\end{Verbatim}

\section*{1.4 Speed Considerations}

As already mentioned in the introduction there are some speed considerations implied if you choose to define macros via \texttt{expkvcs}. However the overhead isn't the factor which should hinder you to use \texttt{expkvcs} if you found a reasonable use case. The key-parsing is still faster than with most other \langle \texttt{key} \rangle=\langle \texttt{value} \rangle packages (see the “Comparisons” subsection in the \texttt{expkvcs} documentation).

The speed considerations in this subsection use the first example in this documentation as the benchmark. So we have seven keys and a short sentence which should be
typeset. For comparisons I use the following equivalent \texttt{expkv-def} definitions. Each result is the average between changing no keys from their initial values and altering four. Furthermore I’ll compare three variants of \texttt{expkv-def} with the \texttt{expkv-def} definitions, namely the split example from above, a hash variant using \texttt{ekvcValue} and a hash variant using \texttt{ekvcValueFast}.

\usepackage{expkv-def}
\ekvdefinekeys{keys}
\%
,store name = \KEYSname
,initial name = John Doe
,store age = \KEYSage
,initial age = any
,store nationality = \KEYSnationality
,initial nationality = the Universe
,store hobby = \KEYShobby
,initial hobby = to exist
,store type = \KEYSstepe
,initial type = Mister
,store pronoun = \KEYSpronoun
,initial pronoun = He
,store possessive = \KEYSpossessive
,initial possessive = his
}
\newcommand{\KEYS[1]}{\%
\begingroup
\ekvset{keys}{#1}\%
\KEYSname is a \KEYSstepe from \KEYSnationality. \KEYSpronoun is of \KEYSage age and \KEYSpossessive hobby is \KEYShobby.\%
\endgroup
}
\%
The first comparison removes the typesetting part from all the definitions, so that only the key parsing is compared. In this comparison the \texttt{ekvcValue} and \texttt{ekvcValueFast} variants will not differ, as they are exactly the same until the key usage. We find that the split approach is 1.4 times slower than the \texttt{expkv-def} setup and the hash variants end up in the middle at 1.17 times slower.

Next we put the typesetting part back in. Every call of the macros will typeset the sentences into a box register in horizontal mode. With the typesetting part (which includes the accessing of values) the fastest remains the \texttt{expkv-def} definitions, but split is close at 1.16 times slower, followed by the hash variant with fast accesses at 1.36 times slower, and the safe hash access variant ranks in the slowest 1.8 times slower than \texttt{expkv-def}.

Just in case you’re wondering now, a simple macro taking seven arguments is 30 to 40 times faster than any of those in the argument grabbing and \texttt{\langle key\rangle=\langle value\rangle} parsing part and only 1.5 to 2.8 times faster if the typesetting part is factored in. So the real choke isn’t the parsing.

So to summarize this, if you have a reasonable use case for expandable \texttt{\langle key\rangle=\langle value\rangle} parsing macros you should go on and define them using \texttt{expkv-kv}. If you have a reasonable use case for \texttt{\langle key\rangle=\langle value\rangle} parsing macros but defining them expandable isn’t
necessary for your use you should take advantage of the greater flexibility of non-
expandable \texttt{\langle key\rangle=\langle value\rangle} setups (but if you're after maximum speed there aren't that
many \texttt{\langle key\rangle=\langle value\rangle} parsers beating \texttt{expkvcs}). And if you are after maximum per-
formance maybe ditching the \texttt{\langle key\rangle=\langle value\rangle} interface altogether is a good idea, but
depending on the number of arguments your interface might get convoluted.

1.5 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.

1.6 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: \url{https://github.com/Skillmon/tex_expkv-cs}

1.7 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:

\url{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 \expkv we provide a small \LaTeX\ package that sets up things such that we behave nicely on \LaTeX\ packages and files system. It’ll \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 \expkv{} if the package didn’t already do so – since \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 \expkv{} to save us from retyping them.

\begin{verbatim}
\input{expkv}
\end{verbatim}

We make sure that \expkv-cs.tex\ is only input once:

\begin{verbatim}
\expandafter\ifx\csname ekvcVersion\endcsname\relax
  \else
  \expandafter\endinput
\fi
\end{verbatim}

\begin{verbatim}
\\ekvcVersion\\ekvcDate
\end{verbatim}

We’re on our first input, so let’s store the version and date in a macro.

\begin{verbatim}
\def\ekvcVersion{0.2}
\def\ekvcDate{2020-04-05}
\end{verbatim}

(End definition for \ekvcVersion and \ekvcDate. These functions are documented on page 9.)

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 \ekvc@tmp to use \ProvidesFile, else this will expand to a \relax and do no harm.

\begin{verbatim}
\csname ekvc@tmp\endcsname
\end{verbatim}

Store the category code of @ to later be able to reset it and change it to 11 for now.

\begin{verbatim}
\expandafter\chardef\csname ekvc@tmp\endcsname=\catcode'@\0
\end{verbatim}

\ekvc@tmp will be reused later, but we don’t need it to ever store information long-term after \expkv{}cs was initialized.
\ekvc@keycount We’ll need to keep count how many keys must be defined for each macro in the split variants.

\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.

\def\ekvc@long{}
\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.

\protected\long\def\ekvcSplitAndForward#1#2#3%
\{%
  \ekv@ifdefined{\expandafter\@gobble\string#1}%
  \{\ekvc@err@already@defined{#1}%%
  \{\ekvcSplitAndForward@{#1}{#2}{#3}%%
%
\}

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

\ekvcSplitAndForward@ 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.

\protected\long\def\ekvcSplitAndForward@#1#2#3%
\{%
  \edef\ekvc@set{\string#1}%
  \ekvc@SetupSplitKeys{#3}%
  \ekvc@any@long\edef#1##1%
  {%
    \unexpanded{\ekvset}{\ekvc@set}{##1}%
    \unexpanded\expandafter%
    \{\csname ekvc@split@\the\ekvc@keycount\endcsname\%
    \unexpanded\expandafter{\ekvc@initials{}#2}%
  }%
%
\}

(End definition for \ekvcSplitAndForward@.)

\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.

\protected\long\def\ekvcSplit#1#2#3%
\{%
  \ekv@ifdefined{\expandafter\@gobble\string#1}%
  \{\ekvc@err@already@defined{#1}%%
  \%
  \expandafter
  \{\csname ekvc@split@\the\ekvc@keycount\endcscname\%
  \}
\}

\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 \ekvparse.

\protected\long\def\ekvc@SetupSplitKeys#1\percent{
  \ekvc@keycount=0
  \def\ekvc@any@long{}
  \def\ekvc@initials{}
  \ekvparse\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\percent{\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\percent{
  \ifdefined{ekvc@split@p@#1}{\csname ekvc@split@p@#1\endcsname{#2}}
  \ekvc@SetupSplitKeys@c{#1 #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.
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.

The long prefix lets the internals \texttt{\ekvc@long} and \texttt{\ekvc@any@long} to \texttt{\long} so that the key macro will be long.

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

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

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

\ekvcHashAndForward\ekvcHashAndForward works just like \ekvcSplitAndForward.
\protected\long\def\ekvcHashAndForward@#1#2#3%{\ekv@ifdefined{\expandafter\@gobble\string#1}{}{\ekvcHashAndForward@{#1}{#2}{#3}}} (End definition for \ekvcHashAndForward. This function is documented on page 4.)

\ekvcHashAndForward@ This is more or less the same as \ekvcHashAndForward@. 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.
\protect\long\def\ekvcHashAndForward@#1#2#3%  
{%  
\edef\ekvc@set{"string"#1}%  
\ekvc@SetupHashKeys{#3}%  
\ekvc@any@long\edef#1##1%  
{%  
\unexpanded{\ekvset}{\ekvc@set}{##1}%  
\unexpanded{\ekvc@hash@pack@argument}%  
\unexpanded\expandafter{\ekvc@initials\ekvc@stop#2}%  
}%  
}%  
(End definition for \ekvcHashAndForward@.)

\ekvcHash
\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.
\protect\long\def\ekvcHash#1#2#3%  
{%  
\ekv@ifdefined{\expandafter\@gobble\string#1}%  
{\ekvc@err@already@defined{#1}}%  
{%  
\expandafter  
\ekvcHashAndForward@\expandafter#1\csname ekvc@string#1\endcsname{#2}%  
\ekvc@any@long\expandafter\def\csname ekvc@string#1\endcsname##1{#3}  
}%  
}%  
(End definition for \ekvcHash. This function is documented on page 3.)

\ekvc@hash@pack@argument
All this macro does is pack the values into one argument and forward that to the next
macro.
\long\def\ekvc@hash@pack@argument#1\ekvc@stop#2{#2{#1}}
(End definition for \ekvc@hash@pack@argument.)

\ekvc@SetupHashKeys
\ekvc@SetupHashKeys@a
\ekvc@SetupHashKeys@b
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.
\protect\long\def\ekvc@SetupHashKeys#1%  
{%  
\def\ekvc@any@long{}%  
\def\ekvc@initials{}%  
\ekvparse\ekvc@err@value@required\ekvc@SetupHashKeys@a{#1}%  
}%  
\protected\def\ekvc@SetupHashKeys@b#1%  
{%  
\def\ekvc@long{}%  
\ekvc@ifspace{#1}  
{\ekvc@SetupHashKeys@b#1\ekvc@stop}  
{\ekvc@SetupHashKeys@c{#1}}  
}%  
\protected\def\ekvc@SetupHashKeys@b#1 #2\ekvc@stop%  
{%  
\expandafter\ekvc@SetupHashKeys@b\expandafter#1\csname ekvc@string#1\endcsname{#2}%  
\expandafter\ekvc@SetupHashKeys@c{#1}{#2}%  
}%  
(End definition for \ekvc@SetupHashKeys.)
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`.

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

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

\ekvc@hash@p@long Nothing astonishing here either.

```
\protected\def\ekvc@hash@p@long{%\let\ekvc@long\long
\let\ekvc@any@long\long
\let\ekvc@SetupHashKeys@c
}
```

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

\ekvc@setup@hashmacro The safe hash macros will be executed inside of a \texttt{\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.
\protected\def\ekvc@setup@hashmacro#1\%{
  \ekv@ifdefined{ekvc@hash@#1}\%
  \%\begin{group}
  \edef\ekvc@tmp\{
    \long\gdef
    \unexpanded\expafter{{\csname ekvc@fasthash@#1\endcsname}}\%
    \unexpanded\expafter{{\csname ekvc@hashmark@#1\endcsname}}\%
    \unexpanded\expafter{{\csname ekvc@stop\endcsname}}\%
    \%\end{group}
  \end{group}
  \long\gdef
  \unexpanded\expafter{{\csname ekvc@safehash@#1\endcsname}}\%
  \unexpanded\expafter{{\csname ekvc@safehashmark@#1\endcsname}}\%
  \unexpanded\expafter{{\csname ekvc@mark\endcsname}{\ekvc@err@missing@hash{#1} \%\termination}}\%
  \%\long\gdef
  \unexpanded\expafter{{\csname ekvc@fastsplithash@#1\endcsname}}\%
  \unexpanded\expafter{{\csname ekvc@safesplithash@#1\endcsname}}\%
  \unexpanded\expafter{{\csname ekvc@mark\endcsname}{\ekvc@err@missing@hash{#1} \%\termination}}\%
  \%\long\gdef\unexpanded\expafter
  {\csname ekvc@fasthash@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@hashmark@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@stop\endcsname}\%
  \%\long\gdef\unexpanded\expafter
  {\csname ekvc@fastsplithash@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@safesplithash@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@mark\endcsname}{\ekvc@err@missing@hash{#1} \%\termination}}\%
  \%\long\gdef
  \unexpanded\expafter{\csname ekvc@hashmark@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@mark\endcsname}{\ekvc@err@missing@hash{#1} \%\termination}}\%
  \%\long\gdef
  \unexpanded\expafter{\csname ekvc@hashmark@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@mark\endcsname}{\ekvc@err@missing@hash{#1} \%\termination}}\%
  \%\long\gdef\unexpanded\expafter
  {\csname ekvc@fasthash@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@hashmark@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@stop\endcsname}\%
  \%\long\gdef\unexpanded\expafter
  {\csname ekvc@fastsplithash@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@safesplithash@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@mark\endcsname}{\ekvc@err@missing@hash{#1} \%\termination}}\%
  \%\long\gdef
  \unexpanded\expafter{\csname ekvc@hashmark@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@mark\endcsname}{\ekvc@err@missing@hash{#1} \%\termination}}\%
  \%\long\gdef
  \unexpanded\expafter{\csname ekvc@hashmark@#1\endcsname}\%
  \unexpanded\expafter{\csname ekvc@mark\endcsname}{\ekvc@err@missing@hash{#1} \%\termination}}}\%
\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.

\long\def\ekvcValue#1#2{
\romannumeral'\^^@%\ekv@ifdefined{ekvc@safehash@#1}{{\csname ekvc@safehash@#1\endcsname}{#2}}{{\ekvc@err@unknown@hash{#1} }% keep this space}
}

(End definition for \ekvcValue. This function is documented on page 4.)

\ekvcValueFast

To be as fast as possible, this doesn’t test for anything, assuming the user knows best.

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

(End definition for \ekvcValueFast. This function is documented on page 4.)

\ekvcValueSplit

This splits off a single version

\long\def\ekvcValueSplit#1#2#3{
\ekv@ifdefined{ekvc@safesplithash@#1}{{\csname ekvc@safesplithash@#1\endcsname}{#2}{#3}}{{\ekvc@err@unknown@hash{#1}{#3}{#2}}}
}

(End definition for \ekvcValueSplit. This function is documented on page 4.)

\ekvc@safe@found@hash
\ekvc@safe@no@hash

\long\def\ekvc@safe@found@hash#1#2\ekvc@stop#3{
\ekv@ifdefined{ekvc@safe@found@hash@#1}{#3}{#2}{#1}}

\long\def\ekvc@safe@no@hash#1#2\ekvc@mark\ekvc@safe@found@hash\ekvc@stop#3{
\ekv@ifdefined{ekvc@safe@no@hash@#1}{#3}{#2}{#1}}

}
\ekvcValueSplitFast
Again a fast approach which doesn’t provide too many safety measurements. This needs to build the hash function and expand it before passing the results to the next control sequence. The first step only builds the control sequence.

\long\def\ekvcValueSplitFast\#1\#2\%
\%
\csname ekvc@fastsplithash@#1\endcsname#2\ekvc@stop
\)

(End definition for \ekvcValueSplitFast. This function is documented on page 4.)

\ekvcValueSplitFast@a
This step then expands the hash function once and passes the result to \#3 which should be a single control sequence.

\long\def\ekvcValueSplitFast@\#1\#2\#3\%
\%
\expandafter#3\expandafter{#1#2\ekvc@stop}%
\)

(End definition for \ekvcValueSplitFast@a.)

\ekvc@safehash@ \ekvc@fasthash@ \ekvc@safesplithash@ \ekvc@fastsplithash@
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 \ekvcValueFast in three expansion steps. The safe hash variant has to also stop the \romannumeral expansion.

\long\def\ekvc@safehash@\#1{\ekvc@err@empty@hash\@gobble{} }% keep this space
\long\def\ekvc@fasthash@\#1\ekvc@stop{\ekvc@err@empty@hash}
\long\def\ekvc@safesplithash@\#1\#2{\ekvc@err@empty@hash\#2{\#1}{}}
\long\def\ekvc@fastsplithash@\#1\#2{\ekvc@stop\#2{\ekvc@err@empty@hash\#2{\#1}{}}}

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

\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.

\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%
\%

\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%
\%
\ekv@ifdefined{ekvc@p@#1}%
\csname ekvc@p@#1\endcsname%
\%
\ekv@ifdefined{ekvc@t@#1}%
\csname ekvc@t@#1\endcsname%

2.2.1 Secondary Key Types

The prefixes are pretty straightforward again. Just set \texttt{\ekvc@long} and forward to the \texttt{\ekvc@after@ptype} type.

\begin{verbatim}
\protected\def\ekvc@p@long#1{
  \let\ekvc@long\long
  \ekvc@after@ptype#1\ekvc@stop
}
\protected\def\ekvc@after@ptype#1 #2\ekvc@stop{
  \ekv@ifdefined{ekvc@t@#1}{\csname ekvc@t@#1\endcsname{#2}}{
    \ekvc@err@unknown@keytype{#1}\@gobble}\}
\end{verbatim}

(End definition for \texttt{\ekvc@p@long} and \texttt{\ekvc@after@ptype}.)

The \texttt{meta} and \texttt{nmeta} key types use a nested \texttt{\ekvset} to set other keys in the same macro’s \texttt{\langle set\rangle}.

\begin{verbatim}
\protected\def\ekvc@t@meta{
  \edef\ekvc@tmp{\ekvc@set}\expandafter\ekvc@type@meta\expandafter{\ekvc@tmp}\ekvlet\}
\protected\def\ekvc@t@nmeta#1{
  \ekvc@assert@not@long{nmeta #1}\edef\ekvc@tmp{\ekvc@set}\expandafter\ekvc@type@meta\expandafter{}{}\ekvletNoVal{#1}\}
\protected\long\def\ekvc@type@meta#1#2#3#4#5#6{
  #2\def\ekvc@tmp#3{\ekvset{#1}{#6}}\def#4{\ekvc@set{#5}\ekvc@tmp\}
}
\end{verbatim}

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

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{
  \}
\end{verbatim}

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

\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 definition for \ekvc@t@default.)

2.2.2 Helper Macros

\ekvc@ifspace A test which can be reduced to an if-empty by gobbling everything up to the first space.
\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@A\ekv@ifempty@A}

(End definition for \ekvc@ifspace and \ekvc@ifspace@)
2.2.3 Assertions

Some keys don’t want to be long and we have to educate the user, so let’s throw an error if someone wanted these to be long.

```
\long\def\ekvc@assert@not@long#1\{\ifx\ekvc@long\long\ekvc@err@no@long\{#1\}\fi}
```

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

2.2.4 Messages

Boring unexpandable error messages.

```
\protected\def\ekvc@err@toomany#1\{\errmessage{expkv-cs Error: Too many keys for macro \string{#1}}\}
\protected\def\ekvc@err@value@required#1\{\errmessage{expkv-cs Error: Missing value for key \unexpanded{#1}}\}
\protected\def\ekvc@err@missing@type#1\{\errmessage{expkv-cs Error: Missing type for secondary key \unexpanded{#1}}\}
\protected\def\ekvc@err@no@long#1\{\errmessage{expkv-cs Error: prefix 'long' not accepted for \unexpanded{#1}}\}
\protected\def\ekvc@err@already@defined#1\{\errmessage{expkv-cs Error: Macro \string{#1} already defined}\}
\protected\def\ekvc@err@unknown@keytype#1\{\errmessage{expkv-cs Error: Unknown key type \unexpanded{#1}}\}
\protected\def\ekvc@err@unknown@key#1\{\errmessage{expkv-cs Error: Unknown key \unexpanded{#1} for macro \ekvc@set}\}
```

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

\ekvc@err 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\expandafter\expandafter\\csname ! expkv-cs Error:\endcsname\##1.}\
```

22
\def\ekvc@err{\expandafter\ekv@gobbleto@stop}
\def\ekvc@err@\{\expandafter\ekv@gobbleto@stop\}
(End definition for \ekvc@err and \ekvc@err@)

\long\def\ekvc@err@unknown@hash#1{\ekvc@err{unknown hash '#1'}}
\long\def\ekvc@err@missing@hash#1{\ekvc@err{hash '#1' not found}}
\long\def\ekvc@err@empty@hash{\ekvc@err{empty hash}}
(End definition for \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
Index

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
\kvDate ................................. 6, 9, 13, 23
\kvHash ................................. 3, 181
\kvHashAndForward ................. 4, 164
\kvSecondaryKeys .................. 5, 352
\kvSplit ................................. 3, 47
\kvSplitAndForward ......... 3, 29
\kvValue .................................. 4, 318
\kvValueFast ......................... 4, 325
\kvValueSplit ......................... 4, 326
\kvValueSplitFast ................. 4, 340
\kvVersion .............................. 6, 9, 13, 21
\kvifdefined ......................... 410, 426
\kvifdefinedNoVal .................. 416
\kvlet ...................................... 122, 239, 393
\kvletkv ....................... 412
\kvletkvNoVal ....................... 418
\kvletNoVal ....................... 399, 434
\kvparse ............................. 75, 196, 355
\kvsset ......................... 41, 176, 403

L
long ........................................... 5

M
meta ........................................... 5

N
nmeta ........................................... 5

T
\TeX and \LaTeX \TeXe commands:
\kv@beaneto@stop ....................... 491
\kv@ifdefined ...................... 31, 49, 87, 144, 166, 183, 207, 250, 321, 328, 366, 369, 386
\kv@ifemptyQ ......................... 445
\kv@ifemptyQA ......................... 441, 445
\kv@ifemptyQSB ...................... 440, 441
\kv@ifemptyQfalse .................. 441
\kv@name .................................. 432
\kv@stop ................................... 487
\kv@afterQtype ....................... 375
\kv@anyQlong ......................... 27, 39, 64, 73, 128, 174, 188, 194, 245
\kv@assertQnotQlong .............. 397, 408, 428, 447
\kv@defarggobbler ............. 59, 61, 131
\kv@err ......................... 479, 492, 493, 494
\kv@err@pre ......................... 479
\kv@err@alreadyQdefined ....... 32, 50, 167, 184, 448
\kv@err@emptyQhash .............. 348, 349, 350, 351, 492
\kv@err@missingQhash .......... 270, 300, 492
\kv@err@missingQtype ........... 362, 382, 448
\kv@err@noQlong ................. 447, 461
\kv@err@toomany ................. 58, 448
\kv@err@unknownQhash .......... 323, 330, 492
\kv@err@unknownQkey ........... 422, 436, 474
\kv@err@unknownQkeytype ...... 371, 388, 470
\kv@err@valueQrequired ...... 75, 196, 355, 448
\kv@fasthash ......................... 348
\kv@fastsplitQhash ................ 348
\kv@hashQlong ......................... 442
\kv@hash@packQargument ........ 177, 191
\kv@ifspace ........... 81, 201, 360, 377, 438
\kv@ifspace@nothing ............ 438
\kv@initials ............................
\kv@keycount ......................... 26, 43, 54, 57, 61, 72, 79, 99, 103, 118, 123
\kv@long .................. 27, 80, 105, 127, 200, 223, 244, 359, 379, 393, 447
\kv@mark ................................
\kv@p@long .............................. 375
\kv@safeQfoundQhash ............... 296, 332
\kv@safeQnotQhash ................. 300, 332
\kv@safeQhash ......................... 348
\kv@safeQsplitQhash ................ 348
\kv@set ................................
\kv@unstop ......................... 37, 41, 107, 110, 122, 172, 176,
ekvc@setup@hashmacro .................. 240, 248
ekvc@setup@splitmacro ................ 123, 132
ekvc@SetupHashKeys ................. 173, 192
ekvc@SetupHashKeys@a ............ 192
ekvc@SetupHashKeys@b ............ 192
ekvc@SetupHashKeys@c ............ 192

.................... 203, 209, 211, 246
ekvc@SetupSplitKeys ............. 38, 70
ekvc@SetupSplitKeys@a .......... 70
ekvc@SetupSplitKeys@b .......... 70
ekvc@SetupSplitKeys@c .......... 70, 129
ekvc@split@l ...................... 132
ekvc@split@p@long ................. 125
ekvc@stop ......................... 82,
85, 178, 191, 202, 205, 259, 271, 279,

ekvc@t@alias .......................... 406
ekvc@t@default ....................... 424
ekvc@t@meta .......................... 390
ekvc@t@nmeta ......................... 390
ekvc@tmp .................... 2, 55, 66, 94, 97, 107,
108, 121, 122, 131, 133, 140, 147,
160, 214, 217, 225, 226, 238, 239,
253, 314, 392, 393, 398, 399, 403,
404, 409, 413, 419, 422, 429, 434, 495
ekvc@type@meta ...................... 390
ekvcHashAndForward@ ............ 168, 170, 187
ekvcSecondaryKeys@a .......... 355, 357
ekvcSecondaryKeys@b .......... 361, 364
ekvcSplitAndForward@ .......... 33, 35, 53
ekvcValueSplitFast@ ............. 344
ekvcValueSplitFast@a ........... 344