

exp_kvICS

define expandable $\langle key \rangle = \langle value \rangle$ macros using exp_kv

Jonathan P. Spratte*

2020-04-05 v0.2

Abstract

exp_kvICS provides two small interfaces to define expandable $\langle key \rangle = \langle value \rangle$ macros using exp_kv. It therefore lowers the entrance boundary to expandable $\langle key \rangle = \langle value \rangle$ macros. The stylised name is exp_kvICS 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	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 L ^A T _E X 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

*jspratte@yahoo.de

1 Documentation

The `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. `expkvics` 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 `cs` in `expkvics` stands for control sequence, because `def` was already taken by `expkvDEF` and "control sequence" is the term D. E. Knuth used in his *TeXbook* for named commands hence macros (though he also used the term "macro"). So `expkvics` defines control sequences for and with `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 `expkvics`. The interface has some overhead (though it still can be considered fast – check [subsection 1.4](#)) 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 `expkv` or using `expkvDEF` for convenience. Or you resort to another $\langle key \rangle = \langle value \rangle$ interface.

`expkvics` is usable as generic code and as a \LaTeX package. It'll automatically load `expkv` in the same mode as well. To use it, just use one of

```
\usepackage{expkv-cs} % LaTeX
\input expkv-cs      % plainTeX
```

1.1 Define Macros and Primary Keys

All macros defined with `expkvics` have to be previously undefined or have the `\meaning` of `\relax`. This is necessary as there is no way to undefine keys once they are set up (neither `expkv` nor `expkvics` 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 `long` (e.g., `long name=Jonathan P. Spratte`). You only need `long` if the key should be able to take a `\par` token. Note however that long keys are a microscopic grain faster (due to some internals of `expkvics`). Only if at least one of the keys was long the $\langle cs \rangle$ in the following defining macros will be `\long`. For obvious reasons there is no possibility to define a macro or key as `\protected`.

At the moment `expkvcs` 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.

<code>\ekvcSplit</code>	<code>\ekvcSplit<cs>{<primary keys>}{<definition>}</code>
-------------------------	---

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

<code>\ekvcSplitAndForward</code>	<code>\ekvcSplit<cs₁₂</code>
-----------------------------------	--

This defines `<cs1>` to be a macro taking one mandatory argument which should contain a `<key>=<value>` list. You can use as many primary keys as you want with this. The primary keys will be forwarded to `<cs2>` 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 `\ekvcValue`, not the parsing, and for a key access using `\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.

<code>\ekvcHash</code>	<code>\ekvcHash<cs>{<primary keys>}{<definition>}</code>
------------------------	--

This defines `<cs>` to be a macro taking one mandatory argument which should contain a `<key>=<value>` 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 `<definition>`, in which you can access the `<value>` of a `<key>` by using `\ekvcValue{<key>}{#1}`.

<hr/> <hr/>	<code>\ekvcHashAndForward</code>	<code>\ekvcHashAndForward<cs₁><cs₂>{<primary keys>}</code>	This defines $\langle cs_1 \rangle$ 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 $\langle cs_2 \rangle$ is used (so this will provide the key list as a single argument to $\langle cs_2 \rangle$). In the underlying macro you can access the $\langle value \rangle$ of a $\langle key \rangle$ by using <code>\ekvcValue{<key>}{#1}</code> .
<hr/>	<code>\ekvcValue</code>	<code>\ekvcValue{<key>}{<key list>}</code>	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 <code>\ekvcHash</code> or <code>\ekvcHashAndForward</code> . 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.
<hr/>	<code>\ekvcValueFast</code>	<code>\ekvcValueFast{<key>}{<key list>}</code>	This behaves just like <code>\ekvcValue</code> , but <i>without any</i> safety tests. As a result this is about 1.4 times faster <i>but</i> will throw low level T _E X 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.
<hr/>	<code>\ekvcValueSplit</code>	<code>\ekvcValueSplit{<key>}{<key list>}{<next>}</code>	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 <code>\ekvcValueSplit</code> 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 <code>\ekvcValue</code> 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.
<hr/>	<code>\ekvcValueSplitFast</code>	<code>\ekvcValueSplitFast{<key>}{<key list>}{<next>}</code>	This behaves just like <code>\ekvcValueSplit</code> , but it won't run the same tests, hence it is faster but more error prone, just like the relation between <code>\ekvcValue</code> and <code>\ekvcValueFast</code> .

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 EKVC|.

\ekvcSecondaryKeys	\ekvcSecondaryKeys $\langle cs \rangle \{ \langle key \rangle = \langle value \rangle, \dots \}$
---------------------------	---

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 p-type prefixes followed by one t-type prefix. The syntax of $\langle value \rangle$ is dependent on the used t-prefix.

1.2.1 p-type Prefixes

There is only one p-prefix available, which is long.

long	The following key will be defined \long.
-------------	--

1.2.2 t-type Prefixes

If you're familiar with **expkv**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 (long), which will be available in grey (long), and which will be disallowed in red (long). This will be put flush right next to the syntax line.

meta	meta $\langle key \rangle = \{ \langle key \rangle = \langle value \rangle, \dots \}$	long
-------------	--	------

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

nmeta	nmeta $\langle key \rangle = \{ \langle key \rangle = \langle value \rangle, \dots \}$	long
--------------	---	------

An nmeta key is like a meta key, but it doesn't take a value, so the $\langle key \rangle = \langle value \rangle$ list is static.

alias	alias $\langle key \rangle = \langle key_2 \rangle$	long
--------------	--	------

This assigns the definition of $\langle key_2 \rangle$ to $\langle key \rangle$. As a result $\langle key \rangle$ is an alias for $\langle key_2 \rangle$ behaving just the same. Both the value taking and the NoVal version (that's **expkv** slang for a key not accepting a value) will be copied if they are defined when **alias** is used. Of course, $\langle key_2 \rangle$ has to be defined, be it as a primary or secondary one.

default	default $\langle key \rangle = \{ \langle default \rangle \}$	long
----------------	--	------

If $\langle key \rangle$ is a defined value taking key, you can define a NoVal version with this that will behave as if $\langle key \rangle$ was given $\langle default \rangle$ as its $\langle value \rangle$. Note that this doesn't change the initial values of primary keys set at definition time in **\ekvcSplit** and friends. $\langle key \rangle$ 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 `\ekvcSplit`.

```
\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
}
```

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

```
\ekvcSecondaryKeys\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,
  }
}
```

Now we can describe people using

```
\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
}

```

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 $\langle key \rangle = \langle value \rangle$ list.

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

```

\usepackage{xparse}
\makeatletter
\NewExpandableDocumentCommand\ nonsense{O{ } m}{\ nonsense@a{#1}{#2}}
\ekvcHashAndForward\ nonsense@a\ nonsense@b
{
  keyA = A,
  keyB = B,
  keyC = c,
  keyD = d,
}
\newcommand*\ nonsense@b[2]
{%
  \begin{tabular}{l}
    key & A & \ekvcValue{keyA}{#1} \\
    & B & \ekvcValue{keyB}{#1} \\
    & C & \ekvcValue{keyC}{#1} \\
    & D & \ekvcValue{keyD}{#1} \\
    \multicolumn{2}{l}{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 Speed Considerations

As already mentioned in the introduction there are some speed considerations implied if you choose to define macros via `explX`s. However the overhead isn’t the factor which should hinder you to use `explX`s if you found a reasonable use case. The key-parsing is still faster than with most other $\langle key \rangle = \langle value \rangle$ packages (see the “Comparisons” subsection in the `explX` 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 `expkvDEF` definitions. Each result is the average between changing no keys from their initial values and altering four. Furthermore I'll compare three variants of `expkvICS` with the `expkvDEF` definitions, namely the split example from above, a hash variant using `\ekvcValue` and a hash variant using `\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             = \KEYStype
  ,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 \KEYStype\ 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 `\ekvcValue` and `\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 `expkvDEF` 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 `expkvDEF` 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 in the slowest 1.8 times slower than `expkvDEF`.

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 `<key>=<value>` 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 `<key>=<value>` parsing macros you should go on and define them using `expkvICS`. If you have a reasonable use case for `<key>=<value>` parsing macros but defining them expandable isn't

necessary for your use you should take advantage of the greater flexibility of non-expandable $\langle key \rangle = \langle value \rangle$ setups (but if you're after maximum speed there aren't that many $\langle key \rangle = \langle value \rangle$ parsers beating `expkvcs`). And if you are after maximum performance maybe ditching the $\langle key \rangle = \langle value \rangle$ 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: 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 L^AT_EX 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 L^AT_EX Package

Just like for `expkv` we provide a small L^AT_EX package that sets up things such that we behave nicely on L^AT_EX packages and files system. It'll `\input` the generic code which implements the functionality.

```
1 \RequirePackage{expkv}
2 \def\ekvc@tmp
3   {%
4     \ProvidesFile{expkv-cs.tex}%
5     [%
6       \ekvcDate\space v\ekvcVersion\space
7       define expandable key=val macros using expkv%
8     ]%
9   }
10 \input{expkv-cs.tex}
11 \ProvidesPackage{expkv-cs}%
12 [%
13   \ekvcDate\space v\ekvcVersion\space
14   define expandable key=val macros using expkv%
15 ]
```

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.

```
16 \input expkv
    We make sure that expkv-cs.tex is only input once:
17 \expandafter\ifx\csname ekvcVersion\endcsname\relax
18 \else
19   \expandafter\endinput
20 \fi
```

`\ekvcVersion` We're on our first input, so let's store the version and date in a macro.

```
\ekvcDate
21 \def\ekvcVersion{0.2}
22 \def\ekvcDate{2020-04-05}
```

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

If the L^AT_EX 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.

```
23 \csname ekvc@tmp\endcsname
    Store the category code of @ to later be able to reset it and change it to 11 for now.
24 \expandafter\chardef\csname ekvc@tmp\endcsname=\catcode'\@
25 \catcode'\@=11
```

`\ekvc@tmp` will be reused later, but we don't need it to ever store information long-term after `expkvcs` 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
`\ekvc@any@long` 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 {%
```

```
31 \ekv@ifdefined{\expandafter\@gobble\string#1}%
```

```
32 {\ekvc@err@already@defined{#1}}%
```

```
33 {\ekvcSplitAndForward@{#1}{#2}{#3}}%
```

```
34 }
```

(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`.

```
35 \protected\long\def\ekvcSplitAndForward@#1#2#3%
```

```
36 {%
```

```
37 \edef\ekvc@set{\string#1}%
```

```
38 \ekvc@SetupSplitKeys{#3}%
```

```
39 \ekvc@any@long\edef#1#1%
```

```
40 {%
```

```
41 \unexpanded{\ekvset}{\ekvc@set}{##1}%
```

```
42 \unexpanded\expandafter
```

```
43 {\csname ekvc@split@\the\ekvc@keycount\endcsname}%
```

```
44 \unexpanded\expandafter{\ekvc@initials}{#2}%
```

```
45 }%
```

```
46 }
```

(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.

```
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@\expandafter#1\csname ekvc@\string#1\endcsname{#2}%
```

```
54 \ifnum\ekvc@keycount=0
```

```

55         \def\ekvc@tmp##1##{}%
56     \else
57         \ifnum\ekvc@keycount>9
58             \ekvc@err@toomany{#1}%
59             \ekvc@defarggobbler9%
60         \else
61             \expandafter\ekvc@defarggobbler\the\ekvc@keycount
62         \fi
63     \fi
64     \ekvc@any@long\expandafter
65     \def\csname ekvc@\string#1\expandafter\endcsname
66         \ekvc@tmp##1##2##3##4##5##6##7##8##9%
67         {#3}%
68     }%
69 }

```

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

```

\ekvc@SetupSplitKeys
\ekvc@SetupSplitKeys@a
\ekvc@SetupSplitKeys@b
\ekvc@SetupSplitKeys@c

```

These macros parse the list of keys and set up the key macros. First we need to initialise some macros and start \ekvparsed.

```

70 \protected\long\def\ekvc@SetupSplitKeys#1%
71 {%
72     \ekvc@keycount=0
73     \def\ekvc@any@long{}%
74     \def\ekvc@initials{}%
75     \ekvparsed\ekvc@err@value@required\ekvc@SetupSplitKeys@a{#1}%
76 }

```

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.

```

77 \protected\def\ekvc@SetupSplitKeys@a#1%
78 {%
79     \advance\ekvc@keycount1
80     \def\ekvc@long{}%
81     \ekvc@ifspace{#1}%
82     {\ekvc@SetupSplitKeys@b#1\ekvc@stop}%
83     {\ekvc@SetupSplitKeys@c{#1}}%
84 }

```

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.

```

85 \protected\def\ekvc@SetupSplitKeys@b#1 #2\ekvc@stop
86 {%
87     \ekvc@ifdefined{ekvc@split@p@#1}%
88     {\csname ekvc@split@p@#1\endcsname{#2}}%
89     {\ekvc@SetupSplitKeys@c{#1 #2}}%
90 }

```

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.

```

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

```

```

92  {%
93  \begingroup
94  \edef\ekvc@tmp
95  {%
96  \endgroup
97  \long\def\unexpanded{\ekvc@tmp}####1####2%
98  \unexpanded\expandafter
99  {\csname ekvc@splitmark@the\ekvc@keycount\endcsname}####3%
100  {%
101  ####2%
102  \unexpanded\expandafter
103  {\csname ekvc@splitmark@the\ekvc@keycount\endcsname}{####1}%
104  }%

```

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.

```

105  \unless\ifx\ekvc@long\long
106  \let\unexpanded\expandafter
107  {\csname ekvc@\ekvc@set{#1}\endcsname\ekvc@tmp}%
108  \def\unexpanded{\ekvc@tmp}####1%
109  {%
110  \unexpanded\expandafter{\csname ekvc@\ekvc@set{#1}\endcsname}%
111  {####1}%
112  }%
113  \fi
114  \def\unexpanded{\ekvc@initials}%
115  {%
116  \unexpanded\expandafter{\ekvc@initials}%
117  \unexpanded\expandafter
118  {\csname ekvc@splitmark@the\ekvc@keycount\endcsname{#2}}%
119  }%
120  }%
121  \ekvc@tmp
122  \ekvlet\ekvc@set{#1}\ekvc@tmp
123  \expandafter\ekvc@setup@splitmacro\expandafter{\the\ekvc@keycount}%
124  }

```

(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.

```

125  \protected\def\ekvc@split@p@long
126  {%
127  \let\ekvc@long\long
128  \let\ekvc@any@long\long
129  \ekvc@SetupSplitKeys@c
130  }

```

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

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

```

131  \protected\def\ekvc@defarggobbler#1{\def\ekvc@tmp##1#1##2##{##1#1}}

```

(End definition for \ekvc@defarggobbler.)

\ekvc@setup@splitmacro Since the first split macro is different from the others we manually set that one up now.
 \ekvc@split@1 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.

```

132 \begingroup
133 \edef\ekvc@tmp
134 {%
135   \long\gdef\unexpanded\expandafter{\csname ekvc@split@1\endcsname}%
136   \unexpanded\expandafter{\csname ekvc@splitmark@1\endcsname}%
137   ##1##2##3%
138   {##3{##1}##2}%
139 }
140 \ekvc@tmp
141 \endgroup
142 \protected\def\ekvc@setup@splitmacro#1%
143 {%
144   \ekvc@ifdefined{ekvc@split@#1}{}%
145   {%
146     \begingroup
147     \edef\ekvc@tmp
148     {%
149       \long\gdef
150       \unexpanded\expandafter{\csname ekvc@split@#1\endcsname}%
151       ####1%
152       \unexpanded\expandafter{\csname ekvc@splitmark@#1\endcsname}%
153       #####2####3%
154       {%
155         \unexpanded\expandafter
156         {\csname ekvc@split@#1\endcsname\the\numexpr#1-1\relax\endcsname}%
157         #####1{{#####2}#####3}%
158       }%
159     }%
160     \ekvc@tmp
161   \endgroup
162   }%
163 }

```

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

\ekvcHashAndForward \ekvcHashAndForward works just like \ekvcSplitAndForward.

```

164 \protected\long\def\ekvcHashAndForward#1#2#3%
165 {%
166   \ekvc@ifdefined{\expandafter\@gobble\string#1}%
167   {\ekvc@err@already@defined{#1}}%
168   {\ekvcHashAndForward@{#1}{#2}{#3}}%
169 }

```

(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.

```

170 \protected\long\def\ekvcHashAndForward@#1#2#3%
171 {%
172     \edef\ekvc@set{\string#1}%
173     \ekvc@SetupHashKeys{#3}%
174     \ekvc@any@long\edef#1##1%
175     {%
176         \unexpanded{\ekvc@set}{\ekvc@set}{##1}%
177         \unexpanded{\ekvc@hash@pack@argument}%
178         \unexpanded\expandafter{\ekvc@initials\ekvc@stop#2}%
179     }%
180 }

```

(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.

```

181 \protected\long\def\ekvcHash#1#2#3%
182 {%
183     \ekvc@ifdefined{\expandafter\@gobble\string#1}%
184     {\ekvc@err@already@defined{#1}}%
185     {%
186         \expandafter
187         \ekvcHashAndForward@\expandafter#1\csname ekvc@\string#1\endcsname{#2}%
188         \ekvc@any@long\expandafter\def\csname ekvc@\string#1\endcsname##1{#3}%
189     }%
190 }

```

(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.

```

191 \long\def\ekvc@hash@pack@argument#1\ekvc@stop#2{#2{#1}}

```

(End definition for \ekvc@hash@pack@argument.)

\ekvc@SetupHashKeys 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.

\ekvc@SetupHashKeys@a
\ekvc@SetupHashKeys@b

```

192 \protected\long\def\ekvc@SetupHashKeys#1%
193 {%
194     \def\ekvc@any@long{}%
195     \def\ekvc@initials{}%
196     \ekvpars\ekvc@err@value@required\ekvc@SetupHashKeys@a{#1}%
197 }
198 \protected\def\ekvc@SetupHashKeys@a#1%
199 {%
200     \def\ekvc@long{}%
201     \ekvc@ifspace{#1}%
202     {\ekvc@SetupHashKeys@b#1\ekvc@stop}%
203     {\ekvc@SetupHashKeys@c{#1}}%
204 }
205 \protected\def\ekvc@SetupHashKeys@b#1 #2\ekvc@stop
206 {%
207     \ekvc@ifdefined{\ekvc@hash@p@#1}%
208     {\csname ekvc@hash@p@#1\endcsname{#2}}%
209     {\ekvc@SetupHashKeys@c{#1 #2}}%
210 }

```

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`.

```

211 \protected\long\def\ekvc@SetupHashKeys@c#1#2%
212 {%
213   \begingroup
214   \edef\ekvc@tmp
215     {%
216       \endgroup
217       \long\def\unexpanded{\ekvc@tmp}####1###2%
218       \unexpanded\expandafter{\csname ekvc@hashmark@#1\endcsname}###3%
219       {%
220         ###2%
221         \unexpanded\expandafter{\csname ekvc@hashmark@#1\endcsname}{###1}%
222       }%
223     \unless\ifx\ekvc@long\long
224       \let\unexpanded\expandafter
225       {\csname ekvc@\ekvc@set{#1}\endcsname\ekvc@tmp}%
226     \def\unexpanded{\ekvc@tmp}###1%
227     {%
228       \unexpanded\expandafter{\csname ekvc@\ekvc@set{#1}\endcsname}%
229       {###1}%
230     }%
231   \fi
232   \def\unexpanded{\ekvc@initials}%
233   {%
234     \unexpanded\expandafter{\ekvc@initials}%
235     \unexpanded\expandafter{\csname ekvc@hashmark@#1\endcsname{#2}}%
236   }%
237 }%
238 \ekvc@tmp
239 \ekvlet\ekvc@set{#1}\ekvc@tmp
240 \ekvc@setup@hashmacro{#1}%
241 }

```

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

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

```

242 \protected\def\ekvc@hash@p@long
243 {%
244   \let\ekvc@long\long
245   \let\ekvc@any@long\long
246   \ekvc@SetupHashKeys@c
247 }

```

(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.


```

248 \protected\def\ekvc@setup@hashmacro#1%
249 {%
250   \ekvc@ifdefined{ekvc@hash@#1}{}%
251   {%
252     \begingroup
253     \edef\ekvc@tmp
254     {%
255       \long\gdef
256       \unexpanded\expandafter{\csname ekvc@fasthash@#1\endcsname}%
257       ####1%
258       \unexpanded\expandafter{\csname ekvc@hashmark@#1\endcsname}%
259       ####2####3\unexpanded{\ekvc@stop}%
260       {####2}%
261     \long\gdef
262     \unexpanded\expandafter{\csname ekvc@safehash@#1\endcsname}%
263     ####1%
264     {%
265       \unexpanded\expandafter{\csname ekvc@@safehash@#1\endcsname}%
266       ####1\unexpanded{\ekvc@mark}{ }%
267       \unexpanded\expandafter
268       {%
269         \csname ekvc@hashmark@#1\endcsname
270         {\ekvc@err@missing@hash{#1} }%
271         \ekvc@mark{\ekvc@stop
272         }%
273       }%
274     \long\gdef
275     \unexpanded\expandafter{\csname ekvc@@safehash@#1\endcsname}%
276     ####1%
277     \unexpanded\expandafter{\csname ekvc@hashmark@#1\endcsname}%
278     ####2####3\unexpanded{\ekvc@mark}####4####5%
279     \unexpanded{\ekvc@stop}%
280     {%
281       ####4####2%
282     }%
283   \long\gdef\unexpanded\expandafter
284   {\csname ekvc@fastsplithash@#1\endcsname}%
285   ####1%
286   \unexpanded\expandafter{\csname ekvc@hashmark@#1\endcsname}%
287   ####2####3\unexpanded{\ekvc@stop}####4%
288   {%
289     ####4{####1####3}{####2}%
290   }%
291   \long\gdef\unexpanded\expandafter
292   {\csname ekvc@safesplithash@#1\endcsname}####1%
293   {%
294     \unexpanded\expandafter
295     {\csname ekvc@@safesplithash@#1\endcsname}%
296     ####1\unexpanded{\ekvc@mark\ekvc@safe@found@hash}%
297     \unexpanded\expandafter
298     {%
299       \csname ekvc@hashmark@#1\endcsname{}%
300       \ekvc@mark{\ekvc@err@missing@hash{#1}\ekvc@safe@no@hash}%
301       \ekvc@stop

```

```

302         }%
303     }%
304     \long\gdef\unexpanded\expandafter
305     {\csname ekvc@@safesplithash@#1\endcsname}%
306     ####1%
307     \unexpanded\expandafter{\csname ekvc@hashmark@#1\endcsname}%
308     ####2####3\unexpanded{\ekvc@mark}####4####5%
309     \unexpanded{\ekvc@stop}%
310     {%
311     ####4{####2}####1####3\unexpanded{\ekvc@stop}%
312     }%
313     }%
314     \ekvc@tmp
315     \endgroup
316 }%
317 }

```

(End definition for \ekvc@setup@hashmacro.)

\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.

```

318 \long\def\ekvcValue#1#2%
319 {%
320     \romannumeral'\^^@%
321     \ekv@ifdefined{\ekvc@safehash@#1}%
322     {\csname ekvc@safehash@#1\endcsname{#2}}%
323     {\ekvc@err@unknown@hash{#1} }% keep this space
324 }

```

(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.

```

325 \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

```

326 \long\def\ekvcValueSplit#1#2#3%
327 {%
328     \ekv@ifdefined{\ekvc@safesplithash@#1}%
329     {\csname ekvc@safesplithash@#1\endcsname{#2}{#3}}%
330     {\ekvc@err@unknown@hash{#1}#3{#2}}%
331 }

```

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

\ekvc@safe@found@hash

\ekvc@safe@no@hash

```

332 \long\def\ekvc@safe@found@hash#1#2\ekvc@stop#3%
333 {%
334     #3{#2}{#1}%
335 }
336 \long\def\ekvc@safe@no@hash#1#2\ekvc@mark\ekvc@safe@found@hash\ekvc@stop#3%
337 {%
338     #3{#2}{}%
339 }

```

(End definition for \ekvc@safe@found@hash and \ekvc@safe@no@hash.)

\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.

```
340 \long\def\ekvcValueSplitFast#1#2%
341   {%
342     \csname ekvc@fastsplithash@#1\endcsname#2\ekvc@stop
343   }
```

(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.

```
344 \long\def\ekvcValueSplitFast@#1#2#3%
345   {%
346     \expandafter#3\expandafter{#1#2\ekvc@stop}%
347   }
```

(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.

```
348 \long\def\ekvc@safehash@#1{\ekvc@err@empty@hash\@gobble{}} % keep this space
349 \long\def\ekvc@fasthash@#1\ekvc@stop{\ekvc@err@empty@hash}
350 \long\def\ekvc@safesplithash@#1#2{\ekvc@err@empty@hash#2{#1}{}}
351 \long\def\ekvc@fastsplithash@#1\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.

```
352 \protected\long\def\ekvcSecondaryKeys#1#2%
353   {%
354     \edef\ekvc@set{\string#1}%
355     \ekvparse\ekvc@err@value@required\ekvcSecondaryKeys@a{#2}%
356   }
357 \protected\def\ekvcSecondaryKeys@a#1%
358   {%
359     \def\ekvc@long{}%
360     \ekvc@ifspace{#1}%
361       {\ekvcSecondaryKeys@b#1\ekvc@stop}%
362       {\ekvc@err@missing@type{#1}\@gobble}%
363   }
364 \protected\def\ekvcSecondaryKeys@b#1 #2\ekvc@stop
365   {%
366     \ekv@ifdefined{ekvc@p@#1}%
367       {\csname ekvc@p@#1\endcsname}%
368     {%
369       \ekv@ifdefined{ekvc@t@#1}%
370       {\csname ekvc@t@#1\endcsname}%
371     }
```

```

371         {\ekvc@err@unknown@keytype{#1}\@firstoftwo\@gobble}%
372     }%
373     {#2}%
374 }

```

(End definition for `\ekvcSecondaryKeys`. This function is documented on page 5.)

2.2.1 Secondary Key Types

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

```

375 \protected\def\ekvc@p@long#1%
376 {%
377     \ekvc@ifspace{#1}%
378     {%
379         \let\ekvc@long\long
380         \ekvc@after@ptype#1\ekvc@stop
381     }%
382     {\ekvc@err@missing@type{long #1}\@gobble}%
383 }
384 \protected\def\ekvc@after@ptype#1 #2\ekvc@stop
385 {%
386     \ekvc@ifdefined{ekvc@t@#1}%
387     {\csname ekvc@t@#1\endcsname{#2}}%
388     {\ekvc@err@unknown@keytype{#1}\@gobble}%
389 }

```

(End definition for `\ekvc@p@long` and `\ekvc@after@ptype`.)

`\ekvc@t@meta` The meta and nmeta key types use a nested `\ekvset` to set other keys in the same macro's `<set>`.

```

\ekvc@t@nmeta
\ekvc@type@meta
390 \protected\def\ekvc@t@meta
391 {%
392     \edef\ekvc@tmp{\ekvc@set}%
393     \expandafter\ekvc@type@meta\expandafter{\ekvc@tmp}\ekvc@long{##1}\ekvlet
394 }
395 \protected\def\ekvc@t@nmeta#1%
396 {%
397     \ekvc@assert@not@long{nmeta #1}%
398     \edef\ekvc@tmp{\ekvc@set}%
399     \expandafter\ekvc@type@meta\expandafter{\ekvc@tmp}{-}\ekvletNoVal{#1}%
400 }
401 \protected\long\def\ekvc@type@meta#1#2#3#4#5#6%
402 {%
403     #2\def\ekvc@tmp#3{\ekvset{#1}{#6}}%
404     #4\ekvc@set{#5}\ekvc@tmp
405 }

```

(End definition for `\ekvc@t@meta`, `\ekvc@t@nmeta`, and `\ekvc@type@meta`.)

`\ekvc@t@alias` alias just checks whether there is a key and/or NoVal key defined with the target name and `\let` the key to those.

```

406 \protected\def\ekvc@t@alias#1#2%
407 {%

```

```

408 \ekvc@assert@not@long{alias #1}%
409 \let\ekvc@tmp\@firstofone
410 \ekvifdefined\ekvc@set{#2}%
411 {%
412   \ekvletkv\ekvc@set{#1}\ekvc@set{#2}%
413   \let\ekvc@tmp\@gobble
414 }%
415 {}%
416 \ekvifdefinedNoVal\ekvc@set{#2}%
417 {%
418   \ekvletkvNoVal\ekvc@set{#1}\ekvc@set{#2}%
419   \let\ekvc@tmp\@gobble
420 }%
421 {}%
422 \ekvc@tmp{\ekvc@err@unknown@key{#2}}%
423 }

```

(End definition for \ekvc@t@alias.)

\ekvc@t@default 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.

```

424 \protected\long\def\ekvc@t@default#1#2%
425 {%
426   \ekvifdefined\ekvc@set{#1}%
427   {%
428     \ekvc@assert@not@long{default #1}%
429     \edef\ekvc@tmp
430     {%
431       \unexpanded\expandafter
432       {\csname\ekv@name\ekvc@set{#1}\endcsname{#2}}%
433     }%
434     \ekvletNoVal\ekvc@set{#1}\ekvc@tmp
435   }%
436   {\ekvc@err@unknown@key{#1}}%
437 }

```

(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.
\ekvc@ifspace@

```

438 \long\def\ekvc@ifspace#1%
439 {%
440   \ekvc@ifspace@#1 \ekv@ifempty@B
441   \ekv@ifempty@false\ekv@ifempty@A\ekv@ifempty@B\@firstoftwo
442 }
443 \long\def\ekvc@ifspace@#1 % keep this space
444 {%
445   \ekv@ifempty@\ekv@ifempty@A
446 }

```

(End definition for \ekvc@ifspace and \ekvc@ifspace@.)

2.2.3 Assertions

`\ekvc@assert@not@long` 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.

```
447 \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.

```
\ekvc@err@toomany
\ekvc@err@value@required
\ekvc@err@missing@type
\ekvc@err@already@defined
448 \protected\def\ekvc@err@toomany#1%
449 {%
450   \errmessage{expkv-cs Error: Too many keys for macro '\string#1'}%
451 }
452 \protected\def\ekvc@err@value@required#1%
453 {%
454   \errmessage{expkv-cs Error: Missing value for key '\unexpanded{#1}'}%
455 }
456 \protected\def\ekvc@err@missing@type#1%
457 {%
458   \errmessage
459     {expkv-cs Error: Missing type for secondary key '\unexpanded{#1}'}%
460 }
461 \protected\def\ekvc@err@no@long#1%
462 {%
463   \errmessage
464     {expkv-cs Error: prefix 'long' not accepted for '\unexpanded{#1}'}%
465 }
466 \protected\def\ekvc@err@already@defined#1%
467 {%
468   \errmessage{expkv-cs Error: Macro '\string#1' already defined}%
469 }
470 \protected\def\ekvc@err@unknown@keytype#1%
471 {%
472   \errmessage{expkv-cs Error: Unknown key type '\unexpanded{#1}'}%
473 }
474 \protected\def\ekvc@err@unknown@key#1%
475 {%
476   \errmessage
477     {expkv-cs Error: Unknown key '\unexpanded{#1}' for macro '\ekvc@set'}%
478 }
(End definition for \ekvc@err@toomany and others.)
```

`\ekvc@err` We need a way to throw error messages expandably in some contexts.

```
\ekvc@err@
479 \begingroup
480 \edef\ekvc@err
481 {%
482   \endgroup
483   \unexpanded{\long\def\ekvc@err}##1%
484   {%
485     \unexpanded{\expandafter\ekvc@err@\@firstofone}%
486     {\unexpanded\expandafter{\csname ! expkv-cs Error:\endcsname}##1.}%
```

```

487         \unexpanded{\ekv@stop}%
488     }%
489 }
490 \ekvc@err
491 \def\ekvc@err@{\expandafter\ekv@gobbleto@stop}

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

\ekvc@err@unknown@hash And here are the expandable error messages.
\ekvc@err@empty@hash 492 \long\def\ekvc@err@unknown@hash#1{\ekvc@err{unknown hash ‘#1’}}
\ekvc@err@missing@hash 493 \long\def\ekvc@err@missing@hash#1{\ekvc@err{hash ‘#1’ not found}}
494 \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 @.
495 \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	<u>5</u>
D	
default	<u>5</u>
E	
\ekvcDate	<u>6</u> , <u>9</u> , <u>13</u> , <u>21</u>
\ekvcHash	<u>3</u> , <u>181</u>
\ekvcHashAndForward	<u>4</u> , <u>164</u>
\ekvcSecondaryKeys	<u>5</u> , <u>352</u>
\ekvcSplit	<u>3</u> , <u>3</u> , <u>47</u>
\ekvcSplitAndForward	<u>3</u> , <u>29</u>
\ekvcValue	<u>4</u> , <u>318</u>
\ekvcValueFast	<u>4</u> , <u>325</u>
\ekvcValueSplit	<u>4</u> , <u>326</u>
\ekvcValueSplitFast	<u>4</u> , <u>340</u>
\ekvcVersion	<u>6</u> , <u>9</u> , <u>13</u> , <u>21</u>
\ekvifdefined	<u>410</u> , <u>426</u>
\ekvifdefinedNoVal	<u>416</u>
\ekvlet	<u>122</u> , <u>239</u> , <u>393</u>
\ekvletkv	<u>412</u>
\ekvletkvNoVal	<u>418</u>
\ekvletNoVal	<u>399</u> , <u>434</u>
\ekvparse	<u>75</u> , <u>196</u> , <u>355</u>
\ekvset	<u>41</u> , <u>176</u> , <u>403</u>
L	
long	<u>5</u>
M	
meta	<u>5</u>
N	
nmeta	<u>5</u>
T	
T _E X and L ^A T _E X 2 _ε commands:	
\ekv@gobbleto@stop	<u>491</u>
\ekv@ifdefined	<u>31</u> , <u>49</u> , <u>87</u> , <u>144</u> , <u>166</u> , <u>183</u> , <u>207</u> , <u>250</u> , <u>321</u> , <u>328</u> , <u>366</u> , <u>369</u> , <u>386</u>
\ekv@ifempty@	<u>445</u>
\ekv@ifempty@A	<u>441</u> , <u>445</u>
\ekv@ifempty@B	<u>440</u> , <u>441</u>
\ekv@ifempty@false	<u>441</u>
\ekv@name	<u>432</u>
\ekv@stop	<u>487</u>
\ekvc@after@ptype	<u>375</u>
\ekvc@any@long	<u>27</u> , <u>39</u> , <u>64</u> , <u>73</u> , <u>128</u> , <u>174</u> , <u>188</u> , <u>194</u> , <u>245</u>
\ekvc@assert@not@long	<u>397</u> , <u>408</u> , <u>428</u> , <u>447</u>
\ekvc@defarggobbler	<u>59</u> , <u>61</u> , <u>131</u>
\ekvc@err	<u>479</u> , <u>492</u> , <u>493</u> , <u>494</u>
\ekvc@err@	<u>479</u>
\ekvc@err@already@defined	<u>32</u> , <u>50</u> , <u>167</u> , <u>184</u> , <u>448</u>
\ekvc@err@empty@hash	<u>348</u> , <u>349</u> , <u>350</u> , <u>351</u> , <u>492</u>
\ekvc@err@missing@hash	<u>270</u> , <u>300</u> , <u>492</u>
\ekvc@err@missing@type	<u>362</u> , <u>382</u> , <u>448</u>
\ekvc@err@no@long	<u>447</u> , <u>461</u>
\ekvc@err@toomany	<u>58</u> , <u>448</u>
\ekvc@err@unknown@hash	<u>323</u> , <u>330</u> , <u>492</u>
\ekvc@err@unknown@key	<u>422</u> , <u>436</u> , <u>474</u>
\ekvc@err@unknown@keytype	<u>371</u> , <u>388</u> , <u>470</u>
\ekvc@err@value@required	<u>75</u> , <u>196</u> , <u>355</u> , <u>448</u>
\ekvc@fasthash@	<u>348</u>
\ekvc@fastsplithash@	<u>348</u>
\ekvc@hash@p@long	<u>242</u>
\ekvc@hash@pack@argument	<u>177</u> , <u>191</u>
\ekvc@ifspace ...	<u>81</u> , <u>201</u> , <u>360</u> , <u>377</u> , <u>438</u>
\ekvc@ifspace@	<u>438</u>
\ekvc@initials	<u>44</u> , <u>74</u> , <u>114</u> , <u>116</u> , <u>178</u> , <u>195</u> , <u>232</u> , <u>234</u>
\ekvc@keycount	<u>26</u> , <u>43</u> , <u>54</u> , <u>57</u> , <u>61</u> , <u>72</u> , <u>79</u> , <u>99</u> , <u>103</u> , <u>118</u> , <u>123</u>
\ekvc@long	<u>27</u> , <u>80</u> , <u>105</u> , <u>127</u> , <u>200</u> , <u>223</u> , <u>244</u> , <u>359</u> , <u>379</u> , <u>393</u> , <u>447</u>
\ekvc@mark	<u>266</u> , <u>271</u> , <u>278</u> , <u>296</u> , <u>300</u> , <u>308</u> , <u>336</u>
\ekvc@p@long	<u>375</u>
\ekvc@safe@found@hash	<u>296</u> , <u>332</u>
\ekvc@safe@no@hash	<u>300</u> , <u>332</u>
\ekvc@safehash@	<u>348</u>
\ekvc@safesplithash@	<u>348</u>
\ekvc@set	<u>37</u> , <u>41</u> , <u>107</u> , <u>110</u> , <u>122</u> , <u>172</u> , <u>176</u> ,

225, 228, 239, 354, 392, 398, 404, 410, 412, 416, 418, 426, 432, 434, 477	287, 301, 309, 311, 325, 332, 336, 342, 346, 349, 351, 361, 364, 380, 384
\ekvc@setup@hashmacro 240, <u>248</u>	\ekvc@t@alias <u>406</u>
\ekvc@setup@splitmacro 123, <u>132</u>	\ekvc@t@default <u>424</u>
\ekvc@SetupHashKeys 173, <u>192</u>	\ekvc@t@meta <u>390</u>
\ekvc@SetupHashKeys@a <u>192</u>	\ekvc@t@nmeta <u>390</u>
\ekvc@SetupHashKeys@b <u>192</u>	\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@SetupHashKeys@c 203, 209, 211, 246	\ekvc@type@meta <u>390</u>
\ekvc@SetupSplitKeys 38, <u>70</u>	\ekvcHashAndForward@ ... 168, <u>170</u> , 187
\ekvc@SetupSplitKeys@a <u>70</u>	\ekvcSecondaryKeys@a 355, 357
\ekvc@SetupSplitKeys@b <u>70</u>	\ekvcSecondaryKeys@b 361, 364
\ekvc@SetupSplitKeys@c <u>70</u> , 129	\ekvcSplitAndForward@ 33, <u>35</u> , 53
\ekvc@split@1 <u>132</u>	\ekvcValueSplitFast@ 344
\ekvc@split@p@long <u>125</u>	\ekvcValueSplitFast@a <u>344</u>
\ekvc@stop 82, 85, 178, 191, 202, 205, 259, 271, 279,	