Abstract

expkv provides a small interface for \(key\)=\(value\) parsing. The parsing macro is fully expandable, the \(code\) of your keys might be not. expkv is pretty fast, but not the fastest available \(key\)=\(value\) solution (keyval for instance is thrice as fast, but not expandable and it might strip braces it shouldn't have stripped).

Contents

1 Documentation
  1.1 Setting up Keys ................................................. 2
  1.2 Parsing Keys .................................................. 3
  1.3 Miscellaneous ............................................... 4
    1.3.1 Other Macros .............................................. 4
    1.3.2 Bugs .................................................... 5
    1.3.3 Comparisons .............................................. 5
  1.4 Examples ...................................................... 8
    1.4.1 Standard Use-Case ........................................ 8
    1.4.2 An Expandable \(key\)=\(value\) Macro Using \texttt{\textbackslash kvSneak} .......... 8
  1.5 Error Messages .............................................. 10
    1.5.1 Load Time ............................................... 10
    1.5.2 Defining Keys ............................................ 10
    1.5.3 Using Keys ............................................... 10
  1.6 License ....................................................... 11

2 Implementation
  2.1 The \texttt{\textbackslash L\TeX} Package .......................... 12
  2.2 The Generic Code ............................................ 12

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

*jspratte@yahoo.de
1 Documentation

exPkv provides an expandable \langle key \rangle=\langle value \rangle parser. The \langle key \rangle=\langle value \rangle pairs should be given as a comma separated list and the separator between a \langle key \rangle and the associated \langle value \rangle should be an equal sign. Both, the commas and the equal signs, might be of category 12 (other) or 13 (active). To support this is necessary as for example babel turns characters active for some languages, for instance the equal sign is turned active for Turkish.

exPkv is usable as generic code or as a \LaTeX{} package. To use it, just use one of:
\begin{verbatim}
\usepackage{expkv} % LaTeX
\input expkv % plain TeX
\end{verbatim}
The \LaTeX{} package doesn’t do more than expkv.tex, except calling \ProvidesPackage{} and setting things up such that expkv.tex will use \ProvidesFile{}.

1.1 Setting up Keys

exPkv provides a rather simple approach to setting up keys, similar to keyval. However there is an auxiliary package named expkv-def (the files use expkv-def instead because \midline{} isn’t allowed in package names by CTAN) which provides a more sophisticated interface, similar to well established packages like pgfkeys or l3keys.

Keys in exPkv (as in almost all other \langle key \rangle=\langle value \rangle implementations) belong to a set such that different sets can contain keys of the same name. Unlike many other implementations exPkv doesn’t provide means to set a default value, instead we have keys that take values and keys that don’t (the latter are called NoVal keys by exPkv), but both can have the same name (on the user level).

The following macros are available to define new keys. Those macros containing “\texttt{def}” in their name can be prefixed by anything allowed to prefix \def{}, prefixes allowed for \let{} can prefix those with “\texttt{let}” in their name, accordingly. Neither \texttt{set} nor \texttt{key} are allowed to be empty for new keys and must not contain a \texttt{par} or tokens that expand to it – they must be legal inside of \texttt{\csname \dots \endcsname}.

\begin{verbatim}
\ekvdef\ekvdefNoVal\ekvlet\ekvletNoVal
\end{verbatim}

\ekvdef\ekvdefNoVal\ekvlet\ekvletNoVal
\begin{verbatim}
\ekvdef\{\langle set \rangle}\{\langle key \rangle}\{\langle code \rangle}\n\ekvdefNoVal\{\langle set \rangle}\{\langle key \rangle}\{\langle code \rangle}\n\ekvlet\{\langle set \rangle}\{\langle key \rangle}\{\langle cs \rangle}\n\ekvletNoVal\{\langle set \rangle}\{\langle key \rangle}\{\langle cs \rangle}\n\end{verbatim}

Defines a \langle key \rangle taking a value in a \langle set \rangle to expand to \langle code \rangle. In \langle code \rangle you can use \#1 to refer to the given value.

Defines a no value taking \langle key \rangle in a \langle set \rangle to expand to \langle code \rangle.

Let the value taking \langle key \rangle in \langle set \rangle to \langle cs \rangle, there are no checks on \langle cs \rangle enforced.

Let the no value taking \langle key \rangle in \langle set \rangle to \langle cs \rangle, it is not checked whether \langle cs \rangle exists or that it takes no parameter.
Let the \langle key \rangle in \langle set \rangle to \langle key2 \rangle in \langle set2 \rangle, it is not checked whether that second key exists.

Let the \langle key \rangle in \langle set \rangle to \langle key2 \rangle in \langle set2 \rangle, it is not checked whether that second key exists.

\subsection{Parsing Keys}

Splits \langle key = value \rangle pairs on commas. From both \langle key \rangle and \langle value \rangle up to one space is stripped from both ends, if then only a braced group remains the braces are stripped as well. So \ekvset{foo}{bar=baz} and \ekvset{foo}{ \{bar= \{baz\} } will both do \( \langle foobarcode \rangle \langle baz \rangle \), so you can hide commas, equal signs and spaces at the ends of either \langle key \rangle or \langle value \rangle by putting braces around them. If you omit the equal sign the code of the key created with the NoVal variants described in subsection 1.1 will be executed. If \langle key = value \rangle contains more than a single unhidden equal sign, it will be split at the first one and the others are considered part of the value. \ekvset should be nestable.

This macro parses the \langle key = value \rangle pairs and provides those list elements which are only keys as the argument to \langle cs1 \rangle, and those which are a \langle key = value \rangle pair to \langle cs2 \rangle as two arguments. It is fully expandable as well and returns the parsed list in \unexpanded, which has no effect outside of an \expanded or \edef context. If you need control over the necessary steps of expansion you can use \expanded around it.

\ekvbreak, \ekvsneak, and \ekvchangeset and their relatives don't work in \ekvparse. It is analogue to expl3's \keyval_parse:NNn, but not with the same parsing rules -- \keyval_parse:NNn throws an error on multiple equal signs per \langle key = value \rangle pair and on empty \langle key \rangle names in a \langle key = value \rangle pair, both of which \ekvparse doesn't deal with.

As a small example:
\ekvparse\handlekey\handlekeyval{foo = bar, key, baz={zzz}}
would expand to
\handlekeyval{foo}{bar}\handlekey\handlekeyval{baz}{zzz}
and afterwards \handlekey and \handlekeyval would have to further handle the \langle key \rangle. There are no macros like these two contained in \exp, you have to set them up yourself if you want to use \ekvparse (of course the names might differ). If you need the results of \ekvparse as the argument for another macro, you should use \expanded as only then the input stream will contain the output above:
\expandafter\handle\expanded{\ekvparse{k}\kv{foo = bar, key, baz={zzz}}}
would expand to
\handle\kv{foo}{bar}\k\kv{baz}{zzz}
1.3 Miscellaneous

1.3.1 Other Macros

\texttt{expkv} provides some other macros which might be of interest.

\begin{verbatim}
\ekvVersion \ekvDate
\end{verbatim}

These two macros store the version and date of the package.

\begin{verbatim}
\ekvfdefined \ekvfdefinedNoVal
\end{verbatim}

These two macros test whether there is a \texttt{\langle key\rangle} in \texttt{\langle set\rangle}. It is false if either a hash table entry doesn’t exist for that key or its meaning is \texttt{relax}.

\begin{verbatim}
\ekvbreak \ekvbreakPreSneak \ekvbreakPostSneak
\end{verbatim}

Gobbles the remainder of the current \texttt{\ekvset} macro and its argument list and reinserts \texttt{(after)}. So this can be used to break out of \texttt{\ekvset}. The first variant will also gobble anything that has been sneaked out using \texttt{\ekvsneak} or \texttt{\ekvsneakPre}, while \texttt{\ekvbreakPreSneak} will put \texttt{(after)} before anything that has been smuggled and \texttt{\ekvbreakPostSneak} will put \texttt{(after)} after the stuff that has been sneaked out.

\begin{verbatim}
\ekvsneak \ekvsneakPre
\end{verbatim}

Puts \texttt{(after)} after the effects of \texttt{\ekvset}. The first variant will put \texttt{(after)} after any other tokens which might have been sneaked before, while \texttt{\ekvsneakPre} will put \texttt{(after)} before other smuggled stuff. This reads and reinserts the remainder of the current \texttt{\ekvset} macro and its argument list to do its job. A small usage example is shown in subsubsection 1.4.2.

\begin{verbatim}
\ekvchangeset
\end{verbatim}

Replaces the current set with \texttt{(new-set)}, so for the rest of the current \texttt{\ekvset} call, that call behaves as if it was called with \texttt{\ekvset{(new-set)}}. Just like \texttt{\ekvsneak} this reads and reinserts the remainder of the current \texttt{\ekvset} macro to do its job. It is comparable to using \texttt{(key)/.cd} in \texttt{pgfkeys}.

\begin{verbatim}
\ekv@name \ekv@name@set \ekv@name@key
\end{verbatim}

The names of the macros that correspond to a key in a set are build with these macros. The default definition of \texttt{\ekv@name@set} is \texttt{\textbackslash ekv\langle set\rangle("}” and the default of \texttt{\ekv@name@key} is \texttt{“\langle key\rangle"). The complete name is build using \texttt{\ekv@name} which is equivalent to \texttt{\ekv@name@set\{\langle set\}\}	extbackslash ekv@name@key\{\langle key\}\}. For \texttt{NoVal} keys an additional \texttt{N} gets appended irrespective of these macros’ definition, so their name is \texttt{\ekv\langle set\rangle\langle key\\rangle\textbackslash N}. You might redefine \texttt{\ekv@name@set} and \texttt{\ekv@name@key} locally but \texttt{\textbackslash don't redefine \textbackslash ekv@name}!

\footnote{This is a change in behaviour, previously \textit{v0.3} and before \texttt{\textbackslash ekvpars\textbackslash e} would expand in exactly two steps. This isn’t always necessary, but makes the parsing considerably slower. If this is necessary for your application you can put an \texttt{\textbackslash expanded around it and will still be faster since you need only a single \textbackslash expandafter this way.}
1.3.2 Bugs

Just like keyval, exPkv is bug free. But if you find bugs or hidden features\footnote{Thanks, David!} you can tell me about them either via mail (see the first page) or directly on GitHub if you have an account there: https://github.com/Skillmon/tex_expkv

1.3.3 Comparisons

Comparisons of speed are done with a very simple test key and the help of the \texttt{benchmark} package. The key and its usage should be equivalent to
\begin{verbatim}
protected\ekudf{test}{height}{\def\myheight{#1}}
\ekuset{test}{height = 6}
\end{verbatim}

and only the usage of the key, not its definition, is benchmarked. For the impatient, the essence of these comparisons regarding speed and buggy behaviour is contained in Table 1.

As far as I know exPkv is the only fully expandable \texttt{\langle key\rangle}=\texttt{\langle value\rangle} parser. I tried to compare exPkv to every \texttt{\langle key\rangle}=\texttt{\langle value\rangle} package listed on CTAN, however, one might notice that some of those are missing from this list. That's because I didn't get the others to work due to bugs, or because they just provide wrappers around other packages in this list.

In this subsubsection is no benchmark of \texttt{\ekvparse} and \texttt{\keyval_parse:NNn} contained, as most other packages don't provide equivalent features to my knowledge. \texttt{\ekvparse} is slightly faster than \texttt{\ekuset}, but keep in mind that it does less. The same is true for \texttt{\keyval_parse:NNn} compared to \texttt{\keys_set:nn} of expl3 (where the difference is much bigger).

\texttt{keyval} is about two times faster and has a comparable feature set just a slightly different way how it handles keys without values. That might be considered a drawback, as it limits the versatility, but also as an advantage, as it might reduce doubled code. Keep in mind that as soon as someone loads \texttt{xkeyval} the performance of \texttt{keyval} gets replaced by \texttt{xkeyval}'s.

Also \texttt{keyval} has a bug, which unfortunately can’t really be resolved without breaking backwards compatibility for many documents, namely it strips braces from the argument before stripping spaces if the argument isn’t surrounded by spaces, also it might strip more than one set of braces. Hence all of the following are equivalent in their outcome, though the last two lines should result in something different than the first two:
\begin{verbatim}
\setkeys{foo}{{bar=baz}}
\setkeys{foo}{{bar= {baz}}}
\setkeys{foo}{{bar= {  baz}}}
\setkeys{foo}{{bar= {{baz}}}}
\end{verbatim}

\texttt{xkeyval} is roughly fourteen times slower, but it provides more functionality, e.g., it has choice keys, boolean keys, and so on. It contains the same bug as \texttt{keyval} as it has to be compatible with it by design (it replaces \texttt{keyval}'s frontend), but also adds even more cases in which braces are stripped that shouldn’t be stripped, worsening the situation.
ltxkeys is over 300 times slower – which is funny, because it aims to be “[…] faster than these earlier packages [referring to keyval and xkeyval].” Since it aims to have a bigger feature set than xkeyval, it most definitely also has a bigger feature set than exp\textsc{3}. Also, it can’t parse \texttt{long} input, so as soon as your values contain a \texttt{par}, it’ll throw errors. Furthermore, ltxkeys doesn’t strip outer braces at all by design, which, imho, is a weird design choice. In addition ltxkeys loads \texttt{catoptions} which is known to introduce bugs (e.g., see https://tex.stackexchange.com/questions/461783).

\texttt{l3keys} is almost five times slower, but has an, imho, great interface to define keys. It strips all outer spaces, even if somehow multiple spaces ended up on either end. It offers more features, but is pretty much bound to \textsc{expl3} code. Whether that’s a drawback is up to you.

\texttt{pgfkeys} is a bit more than two times slower for one key, but has an \textit{enormous} feature set. However, since adding additional keys doesn’t add as much needed time for \texttt{pgfkeys} compared to \textsc{expl3}, it gets faster than \textsc{expl3} at around eight \texttt{\langle key\rangle=\langle value\rangle} pairs. It has the same or a very similar bug \texttt{keyval} has. The brace bug (and also the category fragility) can be fixed by \texttt{pgfkeyx}, but this package was last updated in 2012 and it slows down \texttt{\pgfkeys} by factor 8. Also I don’t know whether this might introduce new bugs.

\texttt{kvsetkeys} with \texttt{kvdefinekeys} is about three times slower, but it works even if commas and equals have category codes different from 12 (just as some other packages in this list). Else the features of the keys are equal to those of \texttt{keyval}, the parser has more features, though.

\texttt{options} is a bit slower for only a single value, but gets a tad faster than \textsc{expl3} at around 10 \texttt{\langle key\rangle=\langle value\rangle} pairs. It has a much bigger feature set. Unfortunately it also suffers from the premature unbracing bug \texttt{keyval} has.

\texttt{simplekv} is hard to compare because I don’t speak French (so I don’t understand the documentation) and from what I can see, there is no direct way to define the equivalent test key. Nevertheless, I tested the closest possible equivalent of my test key while siding for \texttt{simplekv}’s design not forcing something into it it doesn’t seem to be designed for. It is more than five times slower and has hard to predict behaviour regarding brace and space stripping, similar to \texttt{keyval}. The tested definition was:

\begin{verbatim}
\usepackage{simplekv}
\setKVdefault[simplekv]{height={ abc}} % key setup
\setKV[simplekv]{ height = 6 } % benchmarked
\end{verbatim}

\texttt{yax} is over eighteen times slower. It has a pretty strange syntax, imho, and again a direct equivalent is hard to define. It has the premature unbracing bug, too. Also somehow loading \texttt{yax} broke \texttt{options} for me. The tested definition was:

\begin{verbatim}
\usepackage[yax]
\defactiveparameter yax \{storevalue\myheight yax:height \} % key setup
\setParameterlist[yax]{ height = 6 } % benchmarked
\end{verbatim}
Table 1: Comparison of \textlangle key\textrangle=\textlangle value\textrangle packages. The packages are ordered from fastest to slowest for one \textlangle key\textrangle=\textlangle value\textrangle pair. Benchmarking was done using \texttt{l3benchmark} and the scripts in the \texttt{Benchmarks} folder of the \texttt{git repository}. The columns $p_i$ are the polynomial coefficients of a linear fit to the run-time, $p_0$ can be interpreted as the overhead for initialisation and $p_1$ the cost per key. The $T_0$ column is the actual mean ops needed for an empty list argument, as the linear fit doesn’t match that point well in general. The column “BB” lists whether the parsing is affected by some sort of brace bug, “CF” stands for category code fragile and lists whether the parsing breaks with active commas or equal signs.

<table>
<thead>
<tr>
<th>Package</th>
<th>$p_1$</th>
<th>$p_0$</th>
<th>$T_0$</th>
<th>BB</th>
<th>CF</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyval</td>
<td>13.5</td>
<td>1.8</td>
<td>7.0</td>
<td>yes</td>
<td>yes</td>
<td>2014-10-28</td>
</tr>
<tr>
<td>expkv</td>
<td>26.4</td>
<td>2.4</td>
<td>12.6</td>
<td>no</td>
<td>no</td>
<td>2020-02-22</td>
</tr>
<tr>
<td>options</td>
<td>25.5</td>
<td>10.5</td>
<td>21.4</td>
<td>yes</td>
<td>yes</td>
<td>2015-03-01</td>
</tr>
<tr>
<td>pgfkeys</td>
<td>26.4</td>
<td>40.9</td>
<td>55.8</td>
<td>yes</td>
<td>yes</td>
<td>2020-01-08</td>
</tr>
<tr>
<td>kvsetkeys</td>
<td>*</td>
<td>*</td>
<td>41.6</td>
<td>no</td>
<td>no</td>
<td>2019-12-15</td>
</tr>
<tr>
<td>l3keys</td>
<td>114.5</td>
<td>35.4</td>
<td>52.7</td>
<td>no</td>
<td>no</td>
<td>2020-02-14</td>
</tr>
<tr>
<td>simplekv</td>
<td>160.5</td>
<td>10.8</td>
<td>8.7</td>
<td>yes</td>
<td>yes</td>
<td>2017-08-08</td>
</tr>
<tr>
<td>xkeyval</td>
<td>260.9</td>
<td>180.4</td>
<td>164.2</td>
<td>yes</td>
<td>yes</td>
<td>2014-12-03</td>
</tr>
<tr>
<td>yax</td>
<td>507.6</td>
<td>62.9</td>
<td>123.5</td>
<td>yes</td>
<td>yes</td>
<td>2010-01-22</td>
</tr>
<tr>
<td>ltxkeys</td>
<td>3932.8</td>
<td>4737.8</td>
<td>5883.0</td>
<td>no</td>
<td>no</td>
<td>2012-11-17</td>
</tr>
</tbody>
</table>

*For kvsetkeys the linear model used for the other packages is a poor fit, kvsetkeys seems to have approximately quadratic run-time, the coefficients of the second degree polynomial fit are $p_2 = 9.9, p_1 = 40.5$, and $p_0 = 61.5$. Of course the other packages might not really have linear run-time, but at least from 1 to 20 keys the fits don’t seem too bad (the maximum ratio $p_2/p_1$ for the other packages is $9.7 \times 10^{-3}$). If one extrapolates the fits for 100 \textlangle key\textrangle=\textlangle value\textrangle pairs one finds that most of them match pretty well, the exception being ltxkeys, which behaves quadratic as well with $p_2 = 31.7, p_1 = 3267.4$, and $p_0 = 7177.6$. 


1.4 Examples

1.4.1 Standard Use-Case

Say we have a macro for which we want to create a \{(key)=(value)\} interface. The macro has a parameter, which is stored in the dimension \ourdim having a default value from its initialization. Now we want to be able to change that dimension with the \texttt{width} key to some specified value. For that we'd do

\newdimen\ourdim
\ourdim=150pt
\protected\ekvdef{our}{width}{\ourdim=#1}\relax

as you can see, we use the set our here. We want the key to behave different if no value is specified. In that case the key should not use its initial value, but be smart and determine the available space from \hsize, so we also define

\protected\ekvdefNoVal{our}{width}{\ourdim=.9\hsize}\relax

Now we set up our macro to use this \{(key)=(value)\} interface

\protected\def\ourmacro#1\begingroup\ekvset{our}{#1}\the\ourdim\endgroup

Finally we can use our macro like in the following

\ourmacro{ }\par
\ourmacro{width}\par
\ourmacro{width=5pt}\par

The same key using \texttt{expkv-def} Using \texttt{expkv-def} we can set up the equivalent key using a \{(key)=(value)\} interface, after the following we could use \ourmacro in the same way as above. \texttt{expkv-def} will allocate and initialise \ourdim and define the \texttt{width} key \protected for us, so the result will be exactly the same – with the exception that the default will use \ourdim=.9\hsize\relax instead.

\input expkv-def % or \usepackage{expkv-def}
\ekvdefinekeys{our}{
\dimen width = \ourdim,
qdefault width = .9\hsize,
initial width = 150pt
}

1.4.2 An Expandable \{(key)=(value)\} Macro Using \texttt{ekvsneak}

Let's set up an expandable macro, that uses a \{(key)=(value)\} interface. The problems we’ll face for this are:

1. ignoring duplicate keys
2. default values for keys which weren’t used
3. providing the values as the correct argument to a macro (ordered)
First we need to decide which \texttt{\ekvset} or \texttt{\ekvparse} we want to do this with, \texttt{\ekvset} or \texttt{\ekvparse}. For this example we also want to show the usage of \texttt{\ekvsneak}, hence we'll choose \texttt{\ekvset}. And we'll have to use \texttt{\ekvset} such that it builds a parsable list for our macro internals. To gain back control after \texttt{\ekvset} is done we have to put an internal of our macro at the start of that list, so we use an internal key that uses \texttt{\ekvsneakPre} after any user input.

To ignore duplicates will be easy if the value of the key used last will be put first in the list, so the following will use \texttt{\ekvsneakPre} for the user-level keys. If we wanted some key for which the first usage should be the the binding one we would use \texttt{\ekvsneak} instead for that key.

Providing default values can be done in different ways, we'll use a simple approach in which we'll just put the outcome of our keys if they were used with default values before the parsing list terminator.

Ordering the keys can be done simply by searching for a specific token for each argument which acts like a flag, so our sneaked out values will include specific tokens acting as markers.

Now that we have answers for our technical problems, we have to decide what our example macro should do. How about we define a macro that calculates the sine of a number and rounds that to a specified precision? As a small extra this macro should understand input in radian and degree and the used trigonometric function should be selectable as well. For the hard part of this task (expandably evaluating trigonometric functions) we'll use the \texttt{xfp} package.

First we set up our keys according to our earlier considerations and set up the user-facing macro \texttt{\sine}. The end marker of the parsing list will be a \texttt{\sine@stop} token, which we don't need to define and we put our defaults right before it.

\begin{verbatim}
\RequirePackage{xfp}
\makeatletter
\ekvdef{expex}{f}{\ekvsneakPre{f}{#1}}
\ekvdef{expex}{round}{\ekvsneakPre{\fpeval{\max{\deg}{d}}}{#1}}
\ekvdefNoVal{expex}{degree}{\ekvsneakPre{\deg}{d}}
\ekvdefNoVal{expex}{radian}{\ekvsneakPre{\deg}{d}}
\ekvdefNoVal{expex}{internal}{\ekvsneakPre{\sine@rnd}{\fpeval{\max{\deg}{d}}}}
\newcommand\sine@final{}
\end{verbatim}

\begin{verbatim}
\def\sine@final{\fpeval{\max{\deg}{d}}{\fpeval{\max{\deg}{d}}}{#2}}
\end{verbatim}

For the sake of simplicity we defined the macro \texttt{\sine} with two mandatory arguments, the first being the (key)=(value) list, the second the argument to the trigonometric function. We could’ve used xparse’s facilities here to define an expandable macro which takes an optional argument instead.

Now we need to define some internal macros to extract the value of each key’s last usage (remember that this will be the group after the first special flag-token). For that we use one delimited macro per key.

\begin{verbatim}
\def\sine@final{\fpeval{\max{\deg}{d}}{\fpeval{\max{\deg}{d}}}{#2}}
\end{verbatim}

After the macros \texttt{\sine@r}, \texttt{\sine@de}, and \texttt{\sine@f} the macro \texttt{\sine@f} will see \texttt{\sine@final{\fpeval{\max{\deg}{d}}}{\fpeval{\max{\deg}{d}}}{#2}}. Now \texttt{\sine@final} has to expandably deal with those arguments such that the \texttt{\fpeval} macro of \texttt{xfp} gets the correct
input. Luckily this is pretty straightforward in this example. In \texttt{\fpeval} the trigonometric functions have names such as \texttt{\sin} or \texttt{\cos} and the degree taking variants \texttt{\sind} or \texttt{\cosd}. And since the degree key puts a \texttt{d} in \texttt{#2} and the radian key leaves \texttt{#2} empty all we have to do to get the correct function name is stick the two together.

\newcommand*\sine@final[4]{\fpeval{\round(#1#2(#4),#3)}}
\makeatother

Let's test our macro:

\sine\{\par\}
\sine[round=10]\{\par\}
\sine[f=\cos\,\texttt{radian}][\texttt{pi}]\{\par\}
\edef\myval{\sine[f=tan][1]}\text{\meaning\myval}

\begin{itemize}
  \item 0.866
  \item 0.8660254038
  \item -1
  \item macro:->0.017
\end{itemize}

1.5 Error Messages

\texttt{expkv} should only send messages in case of errors, there are no warnings and no info messages. In this subsection those errors are listed.

1.5.1 Load Time

\texttt{expkv.tex} checks whether $\varepsilon$-\texttt{T}eX is available. If it isn't, an error will be thrown using \texttt{\errmessage}:

\texttt{! expkv Error: \texttt{e-TeX} required.}

1.5.2 Defining Keys

If you get any error from \texttt{expkv} while you’re trying to define a key, the definition will be aborted and gobbled.

If you try to define a key with an empty set name you’ll get:

\texttt{! expkv Error: empty set name not allowed.}

Similarly, if you try to define a key with an empty key name:

\texttt{! expkv Error: empty key name not allowed.}

Both of these messages are done in a way that doesn’t throw additional errors due to \texttt{\global}, \texttt{\long}, etc., not being used correctly if you prefixed one of the defining macros.

1.5.3 Using Keys

This subsubsection contains the errors thrown during \texttt{\ekvset}. The errors are thrown in an expandable manner by providing an undefined macro. In the following messages \texttt{<key>} gets replaced with the problematic key’s name, and \texttt{<set>} with the corresponding set. If any errors during \texttt{⟨key⟩=⟨value⟩} handling are encountered, the entry in the comma separated list will be omitted after the error is thrown and the next \texttt{⟨key⟩=⟨value⟩} pair will be parsed.

If you’re using an undefined key you’ll get:

\texttt{! Undefined control sequence. \texttt{<argument>} \! expkv Error: unknown key (‘<key>’, set ‘<set>’).}
If you’re using a key for which only a normal version and no \NoVal version is defined, but don’t provide a value, you’ll get:

! Undefined control sequence.
<argument> \! expkv Error:

\textit{value required} (‘<key>’, set ‘<set>’).

If you’re using a key for which only a \NoVal version and no normal version is defined, but provide a value, you’ll get:

! Undefined control sequence.
<argument> \! expkv Error:

\textit{value forbidden} (‘<key>’, set ‘<set>’).

If you’re using a set for which you never executed one of the defining macros from subsection 1.1 you’ll get a low level \TeX error, as that isn’t actively tested by the parser (and hence will lead to undefined behaviour and not be gracefully ignored). The error will look like

! Missing \endcsname inserted.
<to be read again>

\! expkv Error: \textit{Set ‘<set>’ undefined}.

1.6 License

Copyright © 2020 Jonathan P. Spratte

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

http://www.latex-project.org/lppl.txt

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

2.1 The \LaTeX\ Package

First we set up the \LaTeX\ package. That one doesn’t really do much except \texttt{\input}ing the generic code and identifying itself as a package.

```latex
\def\ekv@tmp
  {%
    \ProvidesFile{expkv.tex}%
    [\ekvDate\space v\ekvVersion\space an expandable key=val implementation]%
  }
\input{expkv.tex}
\ProvidesPackage{expkv}%
[\ekvDate\space v\ekvVersion\space an expandable key=val implementation]
```

2.2 The Generic Code

The rest of this implementation will be the generic code.

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname numexpr\endcsname\relax
  \errmessage{expkv requires e-\TeX}\
\fi
\csname ekv@tmp\endcsname
\expandafter\chardef\csname ekv@tmp\endcsname=\catcode\@=11
\ekv@tmp
```

```latex
\ekvVersion
\ekvDate
We're on our first input, so lets store the version and date in a macro.
\def\ekvVersion{0.5a}
\def\ekvDate{2020-02-27}
(End definition for \ekvVersion and \ekvDate. These functions are documented on page 4.)
```

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

```latex
\csname ekv@tmp\endcsname
```

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

```
\expandafter\chardef\csname ekv@tmp\endcsname=\catcode\@=11
\ekv@tmp
```

\ekv@tmp might later be reused to gobble any prefixes which might be provided to \texttt{\ekvdef} and similar in case the names are invalid, we just temporarily use it here as means to store the current category code of @ to restore it at the end of the file, we never care for the actual definition of it.
Since branching tests are often more versatile than \if...\else...\fi constructs, we define helpers that are branching pretty fast. Also here are some other utility functions that just grab some tokens. The ones that are also contained in \LaTeX{} don’t use the ekv prefix.

\begin{verbatim}
23 \long\def\@gobble#1{}
24 \long\def\@firstofone#1{#1}
25 \long\def\@firstoftwo#1#2{#1}
26 \long\def\@secondoftwo#1#2{#2}
27 \long\def\ekv@fi@secondoftwo\fi\@firstoftwo#1#2{\fi#2}
28 \long\def\ekv@gobbleto@stop#1\ekv@stop{}\ekv@stop{}
29 \def\ekv@gobble@mark\ekv@mark{}
\end{verbatim}

(End definition for \@gobble and others.)

As you can see \ekv@gobbleto@stop uses a special marker \ekv@stop. The package will use three such markers, the one you’ve seen already, \ekv@mark and \ekv@nil. Contrarily to how for instance expl3 does things, we don’t define them, as we don’t need them to have an actual meaning. This has the advantage that if they somehow get expanded – which should never happen if things work out – they’ll throw an error directly.

We can test for a lot of things building on an if-empty test, so let’s define a really fast one. Since some tests might have reversed logic (true if something is not empty) we also set up macros for the reversed branches.

\begin{verbatim}
30 \long\def\ekv@ifempty#1%{\ekv@ifempty@#1\ekv@ifempty@true}
31 \long\def\ekv@ifempty@#1\ekv@ifempty@true\ekv@ifempty@false\@secondoftwo#1#2{#1}
32 \long\def\ekv@ifempty@true\ekv@ifempty@false\@firstoftwo#1#2{#2}
(End definition for \ekv@ifempty and others.)
\end{verbatim}

The obvious test that can be based on an if-empty is if-blank, meaning a test checking whether the argument is empty or consists only of spaces. Our version here will be tweaked a bit, as we want to check this, but with one leading \ekv@mark token that is to be ignored.

\begin{verbatim}
39 \long\def\ekv@ifblank@\ekv@mark#1{\ekv@ifempty@\ekv@ifempty@true}
40 \long\def\ekv@ifblank@\ekv@mark\ekv@ifempty@true\ekv@ifempty@false\@secondoftwo#1#2{\ekv@ifempty@true}
(End definition for \ekv@ifblank@.)
\end{verbatim}

We’ll need to check whether something is defined quite frequently, so why not define a macro that does this. The following test is expandable, slower than the typical expandable test for undefined control sequences, but faster for defined ones. Since we want to be as fast as possible for correct input, this is to be preferred.

\begin{verbatim}
49 \def\ekv@ifdefined#1{%
50 \expandafter\ifx\csname\ifcsname #1\endcsname #1\else relax\fi\endcsname\relax
51 \ekv@fi\@secondoftwo
(End definition for \ekv@ifdefined.)
\end{verbatim}
Since we can save some time if we only have to create the control sequence once when we know beforehand how we want to use it, we build some other macros for those cases (which we’ll have quite often, once per key usage).

\def\ekv@ifdefined@pair#1#2{% 
\expandafter\ekv@ifdefined@pair@
\csname
\ifcsname #1{#2}\endcsname
 #1{#2}\%
 \else
 relax\%
 \fi
 \fi
\endcsname
}\def\ekv@ifdefined@pair@#1{% 
\ifx#1\relax
\ekv@fi@secondoftwo
\fi
\@firstoftwo
{%\ekv@set@pair@#1\ekv@mark}\}

\def\ekv@ifdefined@key#1#2{% 
\expandafter\ekv@ifdefined@key@
\csname
\ifcsname #1{#2}N\endcsname
 #1{#2}N\%
 \else
 relax\%
 \fi
 \fi
\endcsname
}\def\ekv@ifdefined@key@#1{% 
\ifx#1\relax
\ekv@fi@secondoftwo
\fi
\@firstoftwo#1
}

The keys will all follow the same naming scheme, so we define it here.

\def\ekv@name#1#2{\ekv@name@set{#1}\ekv@name@key{#2}}
\def\ekv@name@set#1{ekv#1(}
\def\ekv@name@key#1{#1)}

(End definition for \ekv@ifdefined@pair and others.)

(End definition for \ekv@name, \ekv@name@set, and \ekv@name@key. These functions are documented on page 4.)
We can misuse the macro name we use to expandably store the set-name in a single token – since this increases performance drastically, especially for long set-names – to throw a more meaningful error message in case a set isn’t defined. The name of \texttt{ekv@undefined@set} is a little bit misleading, as it is called in either case inside of \texttt{csname}, but the result will be a control sequence with meaning \texttt{\relax} if the set is undefined, hence will break the \texttt{csname} building the key-macro which will throw the error message.

\begin{verbatim}
def\ekv@undefined@set#1{! expkv Error: Set '#1' undefined.}
\end{verbatim}

(End definition for \texttt{ekv@undefined@set}.)

\texttt{ekv@checkvalid} We place some restrictions on the allowed names, though, namely sets and keys are not allowed to be empty – blanks are fine (meaning set- or key-names consisting of spaces).

\begin{verbatim}
\protected\def\ekv@checkvalid#1#2{%
  \ekv@ifempty{#1}{}{\def\ekv@tmp{}\errmessage{expkv Error: empty set name not allowed}}%}
  \ekv@ifempty{#2}{}{\def\ekv@tmp{}\errmessage{expkv Error: empty key name not allowed}}% \@secondoftwo
  \@gobble}
\end{verbatim}

(End definition for \texttt{ekv@checkvalid}.)

\texttt{ekvifdefined} \texttt{ekvifdefinedNoVal} And provide user-level macros to test whether a key is defined.

\begin{verbatim}
def\ekvifdefined#1#2{\ekv@ifdefined{\ekv@name{#1}{#2}}} def\ekvifdefinedNoVal#1#2{\ekv@ifdefined{\ekv@name{#1}{#2}N}}
\end{verbatim}

(End definition for \texttt{ekvifdefined} and \texttt{ekvifdefinedNoVal}. These functions are documented on page 4.)

\texttt{ekvdef} \texttt{ekvdefNoVal} \texttt{ekvlet} \texttt{ekvletkv} \texttt{ekvletkvNoVal} Set up the key defining macros \texttt{ekvdef} etc.

\begin{verbatim}
\protected\long\def\ekvdef#1#2#3{%\ekv@checkvalid(#1)(#2)%\expandafter\def\csname ekv@name{#1}{#2}\endcsname##1{#3}\ekv@defset{#1}}
\protected\long\def\ekvdefNoVal#1#2#3{%\ekv@checkvalid(#1)(#2)%\expandafter\def\csname ekv@name{#1}{#2}N\endcsname##1{#3}\ekv@defset{#1}}
\end{verbatim}

15
\protected\def\ekvlet#1#2#3\
\begin{verbatim}
\ekv@checkvalid{#1}{#2}
\expandafter\let\csname\ekv@name{#1}{#2}\endcsname#3
\ekv@defset{#1}
\end{verbatim}
\end{verbatim}
\protected\def\ekvletNoVal#1#2#3\
\begin{verbatim}
\ekv@checkvalid{#1}{#2}
\expandafter\let\csname\ekv@name{#1}{#2}N\endcsname#3
\ekv@defset{#1}
\end{verbatim}
\end{verbatim}
\protected\def\ekvletkv#1#2#3#4\
\begin{verbatim}
\ekv@checkvalid{#1}{#2}
\expandafter\let\csname\ekv@name{#1}{#2}\expandafter\endcsname\csname\ekv@name{#3}{#4}\endcsname
\ekv@defset{#1}
\end{verbatim}
\end{verbatim}
\protected\def\ekvletkvNoVal#1#2#3#4\
\begin{verbatim}
\ekv@checkvalid{#1}{#2}
\expandafter\let\csname\ekv@name{#1}{#2}N\expandafter\endcsname\csname\ekv@name{#3}{#4}N\endcsname
\ekv@defset{#1}
\end{verbatim}
\end{verbatim}

(End definition for \ekvdef and others. These functions are documented on page 2.)

\texttt{\ekv@defset} In order to enhance the speed the set name given to \texttt{\ekvset} will be turned into a control sequence pretty early, so we have to define that control sequence.

\protected\def\ekv@defset#1\
\begin{verbatim}
\expandafter\edef\csname\ekv@undefined@set{#1}\endcsname##1\endcsname\endcsname
\end{verbatim}

(End definition for \ekv@defset.)

\texttt{\ekvset} Set up \texttt{\ekvset}, which should not be affected by active commas and equal signs. The equal signs are a bit harder to cope with and we’ll do that later, but replacing the active commas with commas of category other can be done beforehand. That’s why we define \texttt{\ekvset} here with a temporary meaning just to set up the things with two different category codes. #1 will be a , and #2 will be a =. 

16
\def\ekvset#1#2{% 
  \long\def\ekvset##1##2{% 
    \expandafter\ekv@set\csname\ekv@undefined@set{##1}\endcsname 
    \ekv@mark##2\ekv@stop#1{}% 
  } 
}

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

\ekv@set \ekv@set will split the \langle key\rangle=\langle value\rangle list at active commas. Then it has to check whether there were unprotected other commas and resplit there.

\long\def\ekv@set##1##2#1{% 
  \ekv@gobbleto@markstop##2\ekv@endset\ekv@mark\ekv@stop 
  \ekv@ifhas@eq@other##2=\ekv@ifempty@B\ekv@ifempty@false \ekv@ifempty@A\ekv@ifempty@B\@firstoftwo 
  {\ekv@set@eq@other##1##2\ekv@stop} % 
  \ekv@ifhas@eq@active##2#2\ekv@ifempty@B\ekv@ifempty@false 
  \ekv@ifempty@A\ekv@ifempty@B\@firstoftwo 
  {\ekv@set@eq@active##1##2\ekv@stop} % 
  \ekv@ifblank@##2\ekv@nil\ekv@ifempty@B\ekv@ifempty@true@F 
  \ekv@ifempty@A\ekv@ifempty@B\ekv@ifempty@true@F 
  \ekv@strip{##2}\ekv@set@key##1% 
} 

(End definition for \ekv@set.)

\ekv@endset \ekv@endset is a hungry little macro. It will eat everything that remains of \ekv@set and unbrace the sneaked stuff.

\long\def\ekv@endset 
  \ekv@mark\ekv@stop\ekv@set@other##1,\ekv@stop,\ekv@set##2\ekv@mark 
}{##3} 

(End definition for \ekv@endset.)

\ekv@set@other The macro \ekv@set@other is guaranteed to get only single \langle key\rangle=\langle value\rangle pairs. First we test whether we're done, if not split at equal signs. It is faster to first split at category 12 equal signs and only after that on actives. If there is no equal sign, we need to test whether we got a blank argument and if not this is a NoVal key.

\long\def\ekv@set@other##1##2,% 
{\ekv@gobbleto@markstop##2\ekv@endset@other\ekv@mark\ekv@stop 
  \ekv@ifhas@eq@other##2=\ekv@isempty@B\ekv@isempty@false 
  \ekv@isempty@A\ekv@isempty@B\@firstoftwo 
  {\ekv@set@eq@other##1##2\ekv@stop} % 
  \ekv@ifhas@eq@active##2#2\ekv@isempty@B\ekv@isempty@false 
  \ekv@isempty@A\ekv@isempty@B\@firstoftwo 
  {\ekv@set@eq@active##1##2\ekv@stop} % 
  \ekv@ifblank@##2\ekv@nil\ekv@isempty@B\ekv@isempty@true@F 
  \ekv@isempty@A\ekv@isempty@B\@firstfone 
  {\ekv@strip{##2}\ekv@set@key##1}% 
} 

17
\ekv@set@other \ekv@set@eq@other might not be the correct break point, there might be an active equal sign in the currently parsed key-name. If so, we have to resplit. If the split is correct strip the key-name of outer spaces and braces and feed it to \ekv@set@pair.

\long\def\ekv@set@eq@other##1##2=%
{\ekv@ifhas@eq@active##2#2\ekv@ifempty@B\ekv@ifempty@false
{\ekv@strip{##2}\ekv@set@pair##1}}

\ekv@set@eq@active We need to handle the active equal signs.
\long\def\ekv@set@eq@active##1##2#2%
{\ekv@strip{##2}\ekv@set@pair##1}

\ekv@ifhas@eq@other We have to set up the testing macros for our equal signs and \ekv@endset@other.
\long\def\ekv@ifhas@eq@other\ekv@mark##1={\ekv@ifempty@B\ekv@ifempty@false
\ekv@ifempty@A\ekv@ifempty@B\ekv@firstoftwo
##2%\ekv@set@other##3\ekv@mark}{}

\ekv@set@eq@active And we have to set up the testing macros for our equal signs and \ekv@endset@other.
\long\def\ekv@set@eq@active\ekv@mark##1{\ekv@ifempty@B\ekv@ifempty@false
\ekv@ifempty@A\ekv@ifempty@B\ekv@firstoftwo
##2%
\ekv@set@other##3\ekv@mark}

\ekv@endset@other (End definition for \ekv@set@eq@other.)

\ekv@break\ekv@breakPreSneak\ekv@breakPostSneak Provide macros that can completely stop the parsing of \ekvset, who knows what it’ll be useful for.
\long\def\ekv@break\ekv@breakPreSneak\ekv@breakPostSneak##1##2\ekv@stop#1##3%{##3##1\ekv@stop#1{##3##1}}

\ekv@break\ekv@breakPreSneak\ekv@breakPostSneak (End definition for \ekv@break, \ekv@breakPreSneak, and \ekv@breakPostSneak. These functions are documented on page 4.)

\ekv@sneak\ekv@sneakPre One last thing we want to do for \ekvset is to provide macros that just smuggle stuff after \ekvset’s effects.
\long\def\ekv@sneak\ekv@sneakPre##1##2\ekv@stop#1##3%
{##2\ekv@stop#1{##3##1}}

(End definition for \ekv@sneak, \ekv@sneakPre. These functions are documented on page 4.)
\long\def\ekvsneakPre##1##2\ekv@stop##3\% 
{%
  ##2\ekv@stop##1(##1##3)\%
}
(End definition for \ekvsneak and \ekvsneakPre. These functions are documented on page 4.)

\ekvparsede Additionally to the \ekvset macro we also want to provide an \ekvparsede macro, that
has the same scope as \keyval_parse:NNn from expl3. This is pretty analogue to the
\ekvset implementation, we just put an \unexpanded here and there instead of other
macros to stop the \expanded on our output.
\long\def\ekvparsede##1##2##3\% 
{%
  \ekv@parse##1##2\ekv@mark##3#1\ekv@stop#1\%
}
(End definition for \ekvparsede. This function is documented on page 3.)

\ekvparsede Additionally to the \ekvset macro we also want to provide an \ekvparsede macro, that
has the same scope as \keyval_parse:NNn from expl3. This is pretty analogue to the
\ekvset implementation, we just put an \unexpanded here and there instead of other
macros to stop the \expanded on our output.
\long\def\ekvparsede##1##2##3\% 
{%
  \ekv@parse##1##2\ekv@mark##3#1\ekv@stop#1\%
}
(End definition for \ekvparsede. This function is documented on page 3.)

\ekv@endparse Additionally to the \ekvset macro we also want to provide an \ekvparsede macro, that
has the same scope as \keyval_parse:NNn from expl3. This is pretty analogue to the
\ekvset implementation, we just put an \unexpanded here and there instead of other
macros to stop the \expanded on our output.
\long\def\ekv@endparse 
{%
  \ekv@mark\ekv@stop\ekv@parse@other##1\,\ekv@stop\ekv@parse##2\ekv@mark
}
(End definition for \ekv@endparse.)

\ekvparseother Additionally to the \ekvset macro we also want to provide an \ekvparsede macro, that
has the same scope as \keyval_parse:NNn from expl3. This is pretty analogue to the
\ekvset implementation, we just put an \unexpanded here and there instead of other
macros to stop the \expanded on our output.
\long\def\ekvparseother##1##2##3\% 
{%
  \ekv@gobbleto@markstop##3\ekv@endparse\ekv@mark\ekv@stop
  \ekv@parse@other##1##2##3\ekv@stop\ekv@mark
}
(End definition for \ekvparseother.)
Finally really setting things up with `\ekvset`'s temporary meaning:

```latex
\begingroup
\catcode`\,=13
\catcode`\,=13
\ekvset,=
\endgroup
```
\texttt{\textbackslash ekvchangeset} Provide a macro that is able to switch out the current (\texttt{set}) in \texttt{\ekvset}. This operation is slow (by comparison, it should be slightly faster than \texttt{\ekvsneak}), but allows for something similar to \texttt{pgfkeys}'s (\texttt{key})/\texttt{.cd} mechanism. However this operation is more expensive than \texttt{/cd} as we can't just redefine some token to reflect this, but have to switch out the set expandably, so this works similar to the \texttt{\ekvsneak} macros reading and reinserting the remainder of the (\texttt{key})=(\texttt{value}) list.

\begin{verbatim}
def\ekvchangeset\%  \expandafter\ekv@changeset\csname\ekv@undefined@set{#1}\endcsname\ekv@mark \endverbatim

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

\texttt{\textbackslash ekvchangeset} This macro does the real change-out of \texttt{\ekvchangeset}. We introduced an \texttt{\ekv@mark} to not accidentally remove some braces which we have to remove again.

\begin{verbatim}
\longdef\ekv@changeset\%\#1\#2\ekv@set@other\#3\#4\ekv@set\%  \ekv@gobble@mark\#2\ekv@set@other\#1\#4\ekv@set\% \endverbatim

(End definition for \texttt{\ekvchangeset}.)

\texttt{\textbackslash ekv@gobbleto@markstop} The \texttt{\ekv@gobbleto@markstop} can be used to test for \texttt{\ekv@stop} similar to our if-empty test, but instead of using tokens which are used nowhere else (\texttt{\ekv@ifempty@A} and \texttt{\ekv@ifempty@B}) it uses \texttt{\ekv@mark} and \texttt{\ekv@stop}.

\begin{verbatim}
\longdef\ekv@gobbleto@markstop\%\#1\ekv@mark\ekv@stop\{} \endverbatim

(End definition for \texttt{\ekv@gobbleto@markstop}.)

\texttt{\textbackslash ekv@set@pair} \texttt{\textbackslash ekv@set@pair@} \texttt{\textbackslash ekv@set@pair} gets invoked with the space and brace stripped key-name as its first argument, the set-macro as the second argument, and following that is the key-value right delimited by an \texttt{\ekv@stop}.

\begin{verbatim}
\longdef\ekv@set@pair\%\#1\#2\%  \ekv@ifdefined@pair\%\#2\#1\%  \%  \ekv@ifdefined\%\#2\#1\N\%  \ekv@err@noarg  \ekv@err@unknown \ekv@gobbleto@markstop \% \endverbatim

\texttt{\textbackslash ekv@set@pair@} This branch will be executed if the key is not defined as an argument grabbing one. If so test whether there is a NoVal key of the same name or whether the key is unknown. Throw a meaningful error message and gobble the value.

\begin{verbatim}
\ekv@ifdefined\%\#2\#1\N\% \ekv@err@noarg \ekv@err@unknown \% \ekv@gobbleto@markstop \% \endverbatim

\texttt{\textbackslash ekv@set@pair} \texttt{\textbackslash ekv@set@pair@} will call \texttt{\textbackslash ekv@set@pair@} if the key is correctly defined. This will then grab the value, strip outer spaces and braces from it and feed it to the key-macro. Afterwards \texttt{\textbackslash ekv@set@other} will take control again.

\begin{verbatim}
\longdef\ekv@set@pair\%\#1\#2\ekv@stop \% \ekv@strip\#2\#1\% \endverbatim
\texttt{\texttt{ekv@set@key}} \texttt{Analogous to \texttt{ekv@set@pair}, \texttt{ekv@set@key} lets \texttt{ekv@ifdefined@key} test whether a \texttt{NoVal} key is defined, else it'll throw a meaningful error message. Since we don't have to grab any value \texttt{ekv@ifdefined@key} will invoke the key-macro and we're done here, \texttt{ekv@set@other} will take over again.}

\begin{verbatim}
\long\def\ekv@set@key#1#2{\% 
 {\ekv@ifdefined@key#2{#1} { 
 {\ekv@ifdefined{#2{#1}}
 {\ekv@err@reqval
 \ekv@err@unknown
 #2{#1} }
 } 
}
}
\end{verbatim}

\texttt{(End definition for \texttt{ekv@set@key}.)}

\texttt{\texttt{ekv@err}} \texttt{Since \texttt{ekvset} is fully expandable as long as the code of the keys is (which is unlikely) we want to somehow throw expandable errors, in our case via undefined control sequences.}

\begin{verbatim}
\begingroup
\edef\ekv@err{\% 
 {unexpanded{\long\def\ekv@err}##1} 
 {unexpanded{\expandafter\ekv@err@\@firstofone} 
 {\expandafter\noexpand\csname ! expkv Error:\endcsname ##1.} 
 {\ekv@stop} }
\endgroup
\def\ekv@err@{
\expandafter\ekv@gobbleto@stop}
\end{verbatim}

\texttt{(End definition for \texttt{ekv@err} and \texttt{ekv@err@}.)}

\texttt{\texttt{ekv@err@common}} \texttt{Now we can use \texttt{ekv@err} to set up some error messages so that we can later use those instead of the full strings.}

\begin{verbatim}
\long\def\ekv@err@common #1#2{\expandafter\ekv@err@common@string#2{#1}}
\long\def\ekv@err@common@string #1#2' #3. #4#5{\expandafter\ekv@err@common@string\noexpand\expkv Error:\endcsname \string##1.##2.)\expandafter\unexpanded\noexpand\expkv Stop}}
\end{verbatim}

\texttt{(End definition for \texttt{ekv@err@common and others}.)}

\texttt{\texttt{ekv@strip}} \texttt{Finally we borrow some ideas of expl3's \texttt{l3tl} to strip spaces from keys and values. This \texttt{ekv@strip} also strips one level of outer braces after stripping spaces, so an input of \texttt{\{abc\}} becomes \texttt{abc} after stripping. It should be used with \texttt{#1} prefixed by \texttt{ekv@mark}. Also this implementation at most strips one space from both sides.}

\begin{verbatim}
\long\def\ekv@strip#1{\% 
 {long\def\ekv@strip#1} 
}
\end{verbatim}

22
\{\%
  \ekv@strip@a
  \ekv@nil
  \ekv@mark#1\ekv@nil\}
\ekv@stop
\%
\long\def\ekv@strip@a##1\ekv@mark#1##2\ekv@nil##3%
{\%
  \ekv@strip@b##3##1##2\ekv@nil
}\%
\long\def\ekv@strip@b##3##1#1\ekv@nil
{\%
  \ekv@strip@c##1\ekv@nil
}\%
\long\def\ekv@strip@c\ekv@mark##1\ekv@nil##2\ekv@stop##3%
{\%
  #3{##1}
}\%
\}
\ekv@strip{ }
\%
(End definition for \ekv@strip and others.)
Now everything that's left is to reset the category code of \@.
\catcode`\@=\ekv@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.

E
\ekvbreak .......................... 4, 218
\ekvbreakPostSneak .............. 4, 218
\ekvbreakPreSneak ............... 4, 218
\ekvchangeset ...................... 4, 295
\ekvDate ........................... 4, 4, 8, 18
\ekvdef ........................... 2, 108
\ekvdefNoVal ....................... 2, 108
\ekvdefined ......................... 4, 106
\ekvdefinedNoVal .................. 4, 106
\ekvlet .............................. 2, 108
\ekvletkv .......................... 3, 108
\ekvletkvNoVal .................... 3, 108
\ekvletNoVal ........................ 2, 108
\ekvparseset ......................... 3, 4, 229
\ekvset .............................. 3, 163, 294
\ekvsneak .......................... 4, 221
\ekvsneakPreSneak ................. 4, 221
\ekvVersion .......................... 4, 4, 8, 18

T
\TeX{} and \LaTeX{} \texttt{et} commands:
\@firstofone ........... 23, 38, 192, 254, 335
\@firstoftwo .......... 23, 37, 46, 64, 83, 84, 188, 201, 214, 246, 250, 263, 274
\@gobble ......................... 23, 104
\@secondoftwo ............ 23, 33, 36, 102
\ekv@changeset ........................ 297, 299
\ekv@checkvalid .............................. 89, 110, 118, 126, 134, 142, 151
\ekv@defset .............................. 113, 121, 129, 137, 146, 155, 158
\ekv@endparse ......................... 235, 239
\ekv@endparse@other ............ 244, 271
\ekv@endset ......................... 172, 176
\ekv@endset@other .................. 182, 209
\ekv@err .................................. 329, 343
\ekv@err@ ......................... 329
\ekv@err@common ...................... 342
\ekv@err@common@ ................. 342
\ekv@err@noarg ....................... 309, 342
\ekv@err@reval ....................... 324, 342
\ekv@err@unknown ................. 310, 325, 342
\ekv@if@secondoftwo .............. 23, 44, 62, 81
\ekv@gobble@mark ................... 23, 301
\ekv@gobbleto@markstop ........... 172, 182, 235, 244, 303
\ekv@gobbleto@stop ................... 23, 312, 341
\ekv@ifblank ......................... 39, 191, 253
\ekv@ifdefined .............. 40, 106, 107, 308, 323
\ekv@ifdefined@key ................. 48, 321
\ekv@ifdefined@key@ ............... 48
\ekv@ifdefined@pair ................. 48, 306
\ekv@ifdefined@pair@ ............... 48
\ekv@isempty ......................... 30, 91, 97
\ekv@isempty@ ......................... 30, 91, 209, 310
\ekv@isempty@A .......................... 32, 33, 35, 36, 37, 38, 39, 184, 188, 192, 201, 209, 210, 214, 246, 250, 254, 263, 274
\ekv@isempty@B ........................ 32, 33, 35, 36, 37, 38, 183, 184, 187, 188, 191, 192, 200, 201, 212, 214, 245, 246, 249, 250, 253, 254, 262, 263, 273, 274
\ekv@isempty@false .................. 30, 183, 187, 200, 213, 245, 249, 263, 273
\ekv@isempty@true .................... 30
\ekv@ifhas@eq@other .................. 187, 200, 209, 249, 262
\ekv@ifhas@eq@active .............. 183, 209, 245, 273
\ekv@ifmark .......................... 29, 39, 65, 168, 172, 174, 177, 182, 196, 209, 210, 212, 216, 231, 235, 237, 240, 244, 248, 265, 269, 272, 276, 297, 303, 354, 358, 366
\ekv@name ............................. 4, 85, 106, 107, 112, 120, 125, 146, 147, 153, 154
\ekv@name@key ......................... 4, 85, 161
\ekv@name@set ......................... 4, 85, 161
\ekv@nil .............................. 191, 253, 353, 355, 358, 360, 362, 364, 366
\ekv@parse ......................... 231, 233, 240
\ekv@parse@eq@active ............ 251, 264, 267
\ekv@parse@eq@other ................ 247, 260
\ekv@parse@key ....................... 255, 286
\ekv@parse@other .................... 236, 240, 242, 276
\ekv@parse@pair ..................... 265, 269, 278
\ekv@parse@pair@ ................... 278
\ekv@set .............................. 167, 170, 177, 299, 301

24
\ekv@set@eq@active \ldots 189, 202, 205
\ekv@set@eq@other \ldots 185, 198
\ekv@set@key \ldots 193, 219
\ekv@set@other \ldots 173, 177, 180, 216, 299, 301
\ekv@set@pair \ldots 203, 207, 264
\ekv@set@pair@ \ldots 65, 204
\ekv@stop \ldots 28, 168, 172, 173, 177, 182, 185, 189, 212, 218, 219, 220, 221, 223, 225, 227, 231, 235, 236, 240, 244, 247, 251, 272, 278, 303, 315, 337, 356, 366
\ekv@strip \ldots 193, 203, 207, 255, 265, 269, 280, 317, 347
\ekv@strip@a \ldots 347
\ekv@strip@b \ldots 347
\ekv@strip@c \ldots 347
\ekv@tmp \ldots 1, 93, 99, 372
\ekv@undefined@set \ldots 88, 160, 167, 297