expkv|opt

parse class and package options with expkv

Jonathan P. Spratte[*]

2021-04-04 v0.2

Abstract

expkv|opt provides option parsing for classes and packages in LaTeX 2$_\varepsilon$ based on expkv. Global and local options are parsed individually by different commands. The stylised name is expkv|opt but the files use `expkv-opt`, this is due to CTAN-rules which don't allow | in package names since that is the pipe symbol in *nix shells.

Contents

[*]jspratte@yahoo.de

1 Documentation

The **expkv** family provides at its core a ⟨*key*⟩=⟨*value*⟩ parser and additionally packages, one to conveniently define new keys (**expkv|def**) and another to build expandable ⟨*key*⟩=⟨*value*⟩ taking control sequences (**expkv|cs**). Still missing from the mix was a solution to parse LaTeX 2_ε class and package options, a gap that's hereby filled with **expkv|opt**.

With the 2021-05-01 release of LaTeX 2_ε there were some very interesting changes to the package and class options code. It is now possible to use braces inside the options, and we can access options without them being preprocessed. As a result, some but not all restrictions were lifted from the possible option usage. What will still fail is things that aren't save from an `\edef` expansion. One thing that doesn't work any more is the possibility to parse the unused option list, because that one doesn't contain the full information any more.

expkv|opt will fall back to v0.1 if the kernel is older than 2021-05-01. **expkv|opt** shouldn't place any restrictions on the keys, historic shortcomings of the kernel cannot be helped though, so the supported things vary with the kernel version. The one thing that **expkv|opt** doesn't support, which **expkv** alone would, is active commas. But there is no good reason why a comma could be active in the preamble.

The package can be loaded with

```
\usepackage{expkv-opt}
```

and if you need a specific version you can use LaTeX 2_ε's rollback support, so to load v0.1 explicitly use:

```
\usepackage{expkv-opt}[=v0.1]
```

which will load the latest subversion of v0.1 (this shouldn't be done by a package author, but only by a user on a single-document basis if there are some incompatibilities, which is unlikely) Unlike the other packages in the **expkv** family, **expkv|opt** is only provided as a LaTeX package.

Before reading this documentation you should read **expkv**'s documentation and might want to also read the documentation of **expkv|def**.

1.1 Macros

expkv|opt's behaviour if it encounters a defined or an undefined ⟨*key*⟩ depends on which list is being parsed and whether the current file is a class or not. Of course in every case a defined ⟨*key*⟩'s callback will be invoked but an additional action might be executed. For this reason the rule set of every macro will be given below the short description which list it will parse.

During each of the processing macros the current list element (not processed in any way) is stored within the macro `\CurrentOption`.

`\ekvoProcessLocalOptions`

`\ekvoProcessLocalOptions{⟨set⟩}`

This parses the options which are directly passed to the current class or package for an **expkv** ⟨*set*⟩.

Class: defined *nothing*

undefined add the key to the list of unused global options (if the local option list matches the option list of the main class)

Package: defined *nothing*

undefined throw an error

`\ekvoProcessGlobalOptions`

`\ekvoProcessGlobalOptions{⟨set⟩}`

In LaTeX 2$_\varepsilon$ the options given to `\documentclass` are global options. This macro processes the global options for an expkv ⟨*set*⟩.

Class: defined remove the option from the list of unused global options

undefined *nothing*

Package: defined remove the option from the list of unused global options

undefined *nothing*

`\ekvoProcessUnusedGlobalOptions`

`\ekvoProcessUnusedGlobalOptions{⟨set⟩}`

This does no longer work with LaTeX 2$_\varepsilon$ kernels starting from 2021-05-01, since the handling of the unused option list changed and no longer includes the values. As a result this will throw a warning and else will be ignored.

`\ekvoProcessOptionsList`

`\ekvoProcessOptionsList⟨list⟩{⟨set⟩}`

Process the ⟨*key*⟩=⟨*value*⟩ list stored in the macro ⟨*list*⟩.

Class: defined *nothing*

undefined *nothing*

Package: defined *nothing*

undefined *nothing*

`\ekvoUseUnknownHandlers`

`\ekvoUseUnknownHandlers⟨cs₁⟩⟨cs₂⟩` *or*
`\ekvoUseUnknownHandlers*`

With this macro you can change the action expkv|opt executes if it encounters an undefined ⟨*key*⟩ for the next (and only the next) list processing macro. The macro ⟨cs_1⟩ will be called if an undefined ⟨*key*⟩ without a ⟨*value*⟩ is encountered and get one argument, being the ⟨*key*⟩. Analogous the macro ⟨cs_2⟩ will be called if an undefined ⟨*key*⟩ with a ⟨*value*⟩ was specified. It will get two arguments, the first being the ⟨*key*⟩ and the second the ⟨*value*⟩.

If you use the starred variant, it'll not take further arguments. In this case the undefined handlers defined via `\ekvdefunknown` and `\ekvdefunknownNoVal` in the parsing set get used, and if those aren't available they'll simply do nothing.

`\ekvoVersion`
`\ekvoDate`

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

1.2 Example

Let's say we want to create a package that changes the way footnotes are displayed in LaTeX. For this it will essentially just redefine `\thefootnote` and we'll call this package ex-footnote. First we report back which package we are:

```
\ProvidesPackage{ex-footnote}[2020-02-02 v1 change footnotes]
```

Next we'll need to provide the options we want the package to have.

```
\RequirePackage{color}
\RequirePackage{expkv-opt} % also loads expkv
\ekvdef{ex-footnote}{color}{\def\exfn@color{#1}}
\ekvdef{ex-footnote}{format}{\def\exfn@format{#1}}
```

We can provide initial values just by defining the two macros storing the value.

```
\newcommand*\exfn@color{}
\newcommand*\exfn@format{arabic}
```

Next we need to process the options given to the package. The package should only obey options directly passed to it, so we're only using `\ekvoProcessLocalOptions`:

```
\ekvoProcessLocalOptions{ex-footnote}
```

Now everything that's still missing is actually changing the way footnotes appear:

```
\renewcommand*\thefootnote
  {%
    \ifx\exfn@color\@empty
      \csname\exfn@format\endcsname{footnote}%
    \else
      \textcolor{\exfn@color}{\csname\exfn@format\endcsname{footnote}}%
    \fi
  }
```

So the complete code of the package would look like this:

```
\ProvidesPackage{ex-footnote}[2020-02-02 v1 change footnotes]

\RequirePackage{color}
\RequirePackage{expkv-opt} % also loads expkv

\ekvdef{ex-footnote}{color}{\def\exfn@color{#1}}
\ekvdef{ex-footnote}{format}{\def\exfn@format{#1}}
\newcommand*\exfn@color{}
\newcommand*\exfn@format{arabic}

\ekvoProcessLocalOptions{ex-footnote}

\renewcommand*\thefootnote
  {%
    \ifx\exfn@color\@empty
      \csname\exfn@format\endcsname{footnote}%
    \else
```

```
      \textcolor{\exfn@color}{\csname\exfn@format\endcsname{footnote}}%
    \fi
  }
```

And it could be used with one of the following lines:

```
\usepackage{ex-footnote}
\usepackage[format=fnsymbol]{ex-footnote}
\usepackage[color=green]{ex-footnote}
\usepackage[color=red,format=roman]{ex-footnote}
```

1.3 Bugs

If you happen to find bugs, it'd be great if you let me know. Just write me an email (see the front page) or submit a bug report on GitHub: https://github.com/Skillmon/tex_expkv-opt

1.4 License

2 Implementation

First we check whether the LaTeX 2_ε kernel supports raw options. If it doesn't we check whether a specific version was requested, and if that's not the case we manually run `\pkgcls@parse@date@arg` with the last version that supported non-raw options.

```
\IfFormatAtLeastTF{2021/05/01}
  {}
  {%
    \ifx\pkgcls@targetlabel\@empty
      \ifnum\requestedLaTeXdate=\pkgcls@targetdate
        \pkgcls@parse@date@arg{=v0.1}%
      \fi
    \fi
  }
```

Then we tell LaTeX 2_ε where to find which release so that the package rollback code of LaTeX 2_ε can do its thing.

```
\DeclareRelease{v0.1}{2020/10/10}{expkv-opt-2020-10-10.sty}
\DeclareCurrentRelease{v0.2}{2021/04/04}
```

Start the package with the typical LaTeX standards.

`\ekvoVersion`
`\ekvoDate`

Store the packages version and date in two macros.

```
\newcommand*\ekvoVersion{0.2}
\newcommand*\ekvoDate{2021-04-04}
```

(*End definition for* `\ekvoVersion` *and* `\ekvoDate`*. These functions are documented on page 3.*)

And we report who we are and what we need.

```
\ProvidesPackage{expkv-opt}
  [%
    \ekvoDate\space v\ekvoVersion\space
    parse class and package options with expkv%
  ]
\RequirePackage{expkv}
```

2.1 Loop

`\ekvo@CurrentOption@loop`
`\ekvo@CurrentOption@loop@`
`\ekvo@end@loop`

We'll need some loop which can iterate over a comma separated list. The loop is very basic and only works for commas of category 12. First we insert the delimiters for the actual loop.

```
\protected\long\def\ekvo@CurrentOption@loop#1#2%
  {%
    \ekvo@CurrentOption@loop@#2\ekv@mark#1,\ekv@stop,\ekvo@tail
  }
```

The actual loop checks whether the final element has been read and if so ends the loop. Else blank elements are ignored, `\CurrentOption` is set and the macro which parses the list elements called. Then call the next iteration.

```
\long\def\ekvo@CurrentOption@loop@#1#2,%
  {%
    \ekv@gobble@from@mark@to@stop#2\ekvo@end@loop\ekv@stop
    \ekv@ifblank{#2}%
      {}%
      {%
```

```
        \edef\CurrentOption{\unexpanded\expandafter{\@gobble#2}}%
        #1{#2}%
      }%
    \ekvo@CurrentOption@loop@#1\ekv@mark
  }
\long\def\ekvo@end@loop#1\ekvo@tail{}
```

(*End definition for* `\ekvo@CurrentOption@loop`, `\ekvo@CurrentOption@loop@`, *and* `\ekvo@end@loop`.)

2.2 Tests

`\ekvo@ifx@TF`
`\ekvo@ifx@F`

We'll need branching `\ifx` tests so that user input containing unbalanced TeX ifs doesn't break (at least not because of us, everything else is the fault of LaTeX 2_ε).

```
\def\ekvo@ifx@TF#1#2{\ifx#1#2\ekv@fi@firstoftwo\fi\@secondoftwo}
\def\ekvo@ifx@F#1#2{\ifx#1#2\ekv@fi@gobble\fi\@firstofone}
```

(*End definition for* `\ekvo@ifx@TF` *and* `\ekvo@ifx@F`.)

`\ekvo@do@with@set`
`\ekvo@name`
`\ekvo@setname`

This test checks whether the ⟨`set`⟩ is defined. If it is we store it in `\ekvo@setname` and set `\ekvo@name` to a short cut to get the ⟨*key*⟩'s callback name. Next we execute the code in #2, if the ⟨`set`⟩ isn't defined #2 is gobbled.

```
\protected\def\ekvo@do@with@set#1#2%
  {%
    \ekvifdefinedset{#1}%
      {%
        \expandafter
        \let\expandafter\ekvo@name\csname\ekv@undefined@set{#1}\endcsname
        \def\ekvo@setname{#1}%
        #2%
      }%
      {\ekvo@err@undefined@set{#1}}%
  }
```

(*End definition for* `\ekvo@do@with@set`, `\ekvo@name`, *and* `\ekvo@setname`.)

2.3 Key handlers

expkv|opt uses handlers specifying what happens if a parsed ⟨*key*⟩ is defined or undefined.

`\ekvo@handle@undefined@k@pkg`
`\ekvo@handle@undefined@kv@pkg`

The case for undefined keys in a local list of a package is easy, just throw appropriate errors.

```
\protected\long\def\ekvo@handle@undefined@k@pkg#1%
  {%
    \ekv@ifdefined{\ekvo@name{#1}}%
      {\ekvo@err@value@required{#1}}%
      {\ekvo@err@undefined@key{#1}}%
  }
\def\ekvo@handle@undefined@kv@pkg#1#2%
  {%
    \ekv@ifdefined{\ekvo@name{#1}N}%
      {\ekvo@err@value@forbidden{#1}}%
      {\ekvo@err@undefined@key{#1}}%
  }
```

(*End definition for* `\ekvo@handle@undefined@k@pkg` *and* `\ekvo@handle@undefined@kv@pkg`.)

`\ekvo@addto@unused@one`
`\ekvo@addto@unused@two`
`\ekvo@rmfrom@unused@one`
`\ekvo@rmfrom@unused@two`

These macros will add or remove the `\CurrentOption` to or from the list of unused global options. Since `\ekvo@do@unusedoptionlist` will have some overhead before calling the list changing macro in filtering the current option, we use an optimization here in that we check whether the list is empty before calling the `rmfrom` function.

```
\long\def\ekvo@addto@unused@one#1{\ekvo@do@unusedoptionlist\ekvo@addto@list}
\long\def\ekvo@addto@unused@two#1#2{\ekvo@do@unusedoptionlist\ekvo@addto@list}
\long\def\ekvo@rmfrom@unused@one#1%
  {%
    \ekvo@ifx@F\@unusedoptionlist\@empty
      {\ekvo@do@unusedoptionlist\ekvo@rmfrom@list}%
  }
\long\def\ekvo@rmfrom@unused@two#1#2%
  {%
    \ekvo@ifx@F\@unusedoptionlist\@empty
      {\ekvo@do@unusedoptionlist\ekvo@rmfrom@list}%
  }
```

(*End definition for* `\ekvo@addto@unused@one` *and others.*)

`\ekvo@do@unusedoptionlist`
`\ekvo@prepare@unusedoption`
`\ekvo@prepare@unusedoption@a`
`\ekvo@prepare@unusedoption@b`
`\ekvo@prepare@unusedoption@c`

The way the new $\LaTeX\,2_\varepsilon$ kernel handles the unused option list changed. Now not the entire `\CurrentOption` is listed, but just everything up to the first equals sign, and spaces got zapped, doesn't matter whether the raw option list gets used or not. So we have to zap spaces and remove everything from the first equals sign onwards. The code used here will fail if the current option contains an `\ekv@mark` or `\ekv@stop` before the first equals sign (this seems rather unlikely).

```
\protected\def\ekvo@do@unusedoptionlist#1%
  {%
    \let\ekvo@unpreparedCurrentOption\CurrentOption
    \edef\CurrentOption
      {\expandafter\ekvo@prepare@unusedoption\CurrentOption=\ekv@mark}%
    #1\@unusedoptionlist
    \let\CurrentOption\ekvo@unpreparedCurrentOption
  }
\def\ekvo@prepare@unusedoption{\ekvo@prepare@unusedoption@a\@empty}
\def\ekvo@prepare@unusedoption@a#1%
  {%
    \long\def\ekvo@prepare@unusedoption@a##1=##2\ekv@mark
      {%
        \ekvo@prepare@unusedoption@b##1\ekv@stop
          \ekv@mark\ekvo@prepare@unusedoption@b
          #1\ekv@mark\ekvo@prepare@unusedoption@c
      }%
  }
\ekvo@prepare@unusedoption@a{ }
\long\def\ekvo@prepare@unusedoption@b#1 #2\ekv@mark#3{#3#1#2\ekv@mark#3}
\long\def\ekvo@prepare@unusedoption@c
    #1\ekv@stop
    \ekv@mark\ekvo@prepare@unusedoption@b\ekv@mark\ekvo@prepare@unusedoption@c
  {\unexpanded\expandafter{#1}}
```

(*End definition for* `\ekvo@do@unusedoptionlist` *and others.*)

\ekvo@set@handlers@local
\ekvo@set@handlers@global
\ekvo@set@handlers@list

These macros are boring. They just set up the handlers to respect the rules documented earlier.

```
\protected\def\ekvo@set@handlers@local
  {%
    \ekvo@if@need@handlers
      {%
        \ifx\@currext\@clsextension
          \ifx\@classoptionslist\relax
            \let\ekvo@handle@undefined@k\@gobble
            \let\ekvo@handle@undefined@kv\@gobbletwo
          \else
            \expandafter
            \ifx
                \csname @raw@opt@\@currname.\@currext\endcsname
                \@raw@classoptionslist
              \let\ekvo@handle@undefined@k\ekvo@addto@unused@one
              \let\ekvo@handle@undefined@kv\ekvo@addto@unused@two
            \else
              \let\ekvo@handle@undefined@k\@gobble
              \let\ekvo@handle@undefined@kv\@gobbletwo
            \fi
          \fi
        \else
          \let\ekvo@handle@undefined@k\ekvo@handle@undefined@k@pkg
          \let\ekvo@handle@undefined@kv\ekvo@handle@undefined@kv@pkg
        \fi
      }%
  }
\protected\def\ekvo@set@handlers@global
  {%
    \unless\ifx\@unusedoptionlist\@empty
      \let\ekvo@handle@defined@k\ekvo@rmfrom@unused@one
      \let\ekvo@handle@defined@kv\ekvo@rmfrom@unused@two
    \fi
    \ekvo@if@need@handlers
      {%
        \let\ekvo@handle@undefined@k\@gobble
        \let\ekvo@handle@undefined@kv\@gobbletwo
      }%
  }
\protected\def\ekvo@set@handlers@list
  {%
    \ekvo@if@need@handlers
      {%
        \let\ekvo@handle@undefined@k\@gobble
        \let\ekvo@handle@undefined@kv\@gobbletwo
      }%
  }
```

(*End definition for* \ekvo@set@handlers@local*,* \ekvo@set@handlers@global*, and* \ekvo@set@handlers@list*.*)

\ekvo@if@need@handlers
\ekvo@dont@need@handlers

If the user specifies handlers this macro will be let to \ekvo@dont@need@handlers, which will act like \@gobble and also let it to \@firstofone afterwards.

```
\let\ekvo@if@need@handlers\@firstofone
```

```
\protected\long\def\ekvo@dont@need@handlers#1%
  {%
    \let\ekvo@if@need@handlers\@firstofone
  }%
```

(*End definition for* `\ekvo@if@need@handlers` *and* `\ekvo@dont@need@handlers`.)

We have to set the default for the handlers of defined keys, because they don't necessarily get defined before a list is parsed.

```
\let\ekvo@handle@defined@k\@gobble
\let\ekvo@handle@defined@kv\@gobbletwo
```

2.4 Processing list elements

`\ekvo@process@common` All the key processing frontend macros use the same basic structure. `#1` will be a simple test, deciding whether the list will really be parsed or not, `#3` will be the ⟨*set*⟩, and `#2` will be the individual code of the frontend macro which should be executed if both the test in `#1` is true and the ⟨*set*⟩ is defined.

```
\protected\def\ekvo@process@common#1#2#3%
  {%
    #1{\ekvo@do@with@set{#3}{#2}}%
  }
```

(*End definition for* `\ekvo@process@common`.)

`\ekvo@process@list` This macro only expands the list holding macro and forwards it to the loop macro.

```
\protected\def\ekvo@process@list#1%
  {%
    \expandafter\ekvo@CurrentOption@loop\expandafter{#1}\ekvo@parse
  }
```

(*End definition for* `\ekvo@process@list`.)

`\ekvo@parse` This macro calls internals of `\ekvparse` such that the code splitting at commas isn't executed, else this is equivalent to `\ekvparse\ekvo@set@k\ekvo@set@kv{#1}`.

```
\protected\long\def\ekvo@parse#1%
  {%
    \ekv@eq@other#1\ekv@nil\ekv@mark\ekv@parse@eq@other@a
      =\ekv@mark\ekv@parse@eq@active
    \ekvo@set@k\ekvo@set@kv
    \ekvo@tail
  }
```

(*End definition for* `\ekvo@parse`.)

`\ekvo@set@k`
`\ekvo@set@kv` These two macros check whether the key is defined and if so call the handler for defined keys and execute the key, else the handler for undefined keys is called. They have to clean up a bit of code which is left by `\ekvo@parse`.

```
\protected\def\ekvo@set@k#1#2\ekvo@tail
  {%
    \ekv@ifdefined{\ekvo@name{#1}N}%
      {%
        \ekvo@handle@defined@k{#1}%
        \csname\ekvo@name{#1}N\endcsname
      }%
      {\ekvo@handle@undefined@k{#1}}%
  }
\protected\def\ekvo@set@kv#1#2#3\ekvo@tail
  {%
    \ekv@ifdefined{\ekvo@name{#1}}%
      {%
        \ekvo@handle@defined@kv{#1}{#2}%
        \csname\ekvo@name{#1}\endcsname{#2}%
      }%
      {\ekvo@handle@undefined@kv{#1}{#2}}%
  }
```

(*End definition for* `\ekvo@set@k` *and* `\ekvo@set@kv`.)

2.5 List variable helpers

`\ekvo@addto@list`

This macro is rather simple. If the list to which the `\CurrentOption` should be added is empty we can just let the list to the `\CurrentOption`. Else we have to expand the list once and the `\CurrentOption` once.

```
\protected\def\ekvo@addto@list#1%
  {%
    \ekvo@ifx@TF#1\@empty
      {\let#1\CurrentOption}%
      {%
        \edef#1%
          {%
            \unexpanded\expandafter{#1},%
            \unexpanded\expandafter{\CurrentOption}%
          }%
      }%
  }
```

(*End definition for* `\ekvo@addto@list`.)

`\ekvo@rmfrom@list`
`\ekvo@rmfrom@list@`

This works by looping over every list item and comparing it to `\ekvo@curropt` which stores the real `\CurrentOption`. This is comparatively slow, but works for items containing braces unlike what LaTeX 2_ε does. We could be faster for items not containing braces, though.

```
\protected\def\ekvo@rmfrom@list#1%
  {%
    \ekvo@ifx@F#1\@empty
      {%
        \let\ekvo@tmp@list\@empty
        \let\ekvo@curropt\CurrentOption
        \expandafter\ekvo@CurrentOption@loop\expandafter{#1}\ekvo@rmfrom@list@
        \let\CurrentOption\ekvo@curropt
        \let#1\ekvo@tmp@list
      }%
  }
\protected\long\def\ekvo@rmfrom@list@#1%
  {%
    \ekvo@ifx@F\CurrentOption\ekvo@curropt
      {\ekvo@addto@list\ekvo@tmp@list}%
  }
```

(*End definition for* `\ekvo@rmfrom@list` *and* `\ekvo@rmfrom@list@`*.*)

2.6 Errors

`\ekvo@err@undefined@key`
`\ekvo@err@value@required`
`\ekvo@err@value@forbidden`
`\ekvo@err@undefined@set`

Just some macros to throw errors in the few cases an error has to be thrown.

```
\protected\def\ekvo@err@undefined@key#1%
  {%
    \PackageError{expkv-opt}{Undefined key ‘#1’ in set ‘\ekvo@setname’}{}%
  }
\protected\def\ekvo@err@value@required#1%
  {%
    \PackageError{expkv-opt}%
      {Value required for key ‘#1’ in set ‘\ekvo@setname’}%
      {}%
  }
\protected\def\ekvo@err@value@forbidden#1%
  {%
    \PackageError{expkv-opt}%
      {Value forbidden for key ‘#1’ in set ‘\ekvo@setname’}%
      {}%
  }
\protected\def\ekvo@err@undefined@set#1%
  {%
    \PackageError{expkv-opt}%
      {Undefined set ‘#1’}%
      {The set for which you try to parse options isn’t defined in expkv.}%
  }
```

(*End definition for* `\ekvo@err@undefined@key` *and others.*)

2.7 User Interface

The user interface macros just put together the bits and pieces.

`\ekvoProcessLocalOptions`

```
\protected\def\ekvoProcessLocalOptions
  {%
    \ekvo@process@common
      {\ekv@ifdefined{@raw@opt@\@currname.\@currext}\@firstofone\@gobble}%
      {%
        \ekvo@set@handlers@local
        \expandafter
        \ekvo@process@list\csname @raw@opt@\@currname.\@currext\endcsname
        \AtEndOfPackage{\let\@unprocessedoptions\relax}%
      }%
  }
```

(*End definition for* `\ekvoProcessLocalOptions`*. This function is documented on page 2.*)

`\ekvoProcessGlobalOptions`

```
\protected\def\ekvoProcessGlobalOptions
  {%
    \ekvo@process@common{\ekvo@ifx@F\@classoptionslist\relax}%
      {%
```

```
        \ekvo@set@handlers@global
        \ekvo@process@list\@raw@classoptionslist
        \let\ekvo@handle@defined@k\@gobble
        \let\ekvo@handle@defined@kv\@gobbletwo
      }%
  }
```

(*End definition for* `\ekvoProcessGlobalOptions`. *This function is documented on page 3.*)

`\ekvoProcessUnusedGlobalOptions`

```
\protected\def\ekvoProcessUnusedGlobalOptions#1%
  {%
    \PackageWarning{expkv-opt}%
      {This macro no longer works because of changes in the LaTeX2e kernel.}%
  }
```

(*End definition for* `\ekvoProcessUnusedGlobalOptions`. *This function is documented on page 3.*)

`\ekvoProcessOptionsList`

```
\protected\def\ekvoProcessOptionsList#1%
  {%
    \ekvo@process@common{\ekvo@ifx@F#1\@empty}%
      {%
        \ekvo@set@handlers@list
        \ekvo@process@list#1%
      }%
  }
```

(*End definition for* `\ekvoProcessOptionsList`. *This function is documented on page 3.*)

`\ekvoUseUnknownHandlers`

```
\protected\def\ekvoUseUnknownHandlers
  {\@ifstar\ekvoUseUnknownHandlers@s\ekvoUseUnknownHandlers@n}
\protected\def\ekvoUseUnknownHandlers@s
  {%
    \def\ekvo@handle@undefined@k
      {%
        \ekv@ifdefined{\ekvo@name{}uN}%
          {\csname\ekvo@name{}uN\endcsname}%
          {\@gobble}%
      }%
    \long\def\ekvo@handle@undefined@kv##1##2%
      {%
        \ekv@ifdefined{\ekvo@name{}u}%
          {\csname\ekvo@name{}u\endcsname{##2}{##1}}%
          {}%
      }%
    \let\ekvo@if@need@handlers\ekvo@dont@need@handlers
  }
\protected\def\ekvoUseUnknownHandlers@n#1#2%
  {%
    \let\ekvo@handle@undefined@k#1\relax
    \let\ekvo@handle@undefined@kv#2\relax
    \let\ekvo@if@need@handlers\ekvo@dont@need@handlers
  }
```

(*End definition for* `\ekvoUseUnknownHandlers`*. This function is documented on page 3.*)

All user interface macros should be only used in the preamble.

```
\@onlypreamble\ekvoProcessLocalOptions
\@onlypreamble\ekvoProcessGlobalOptions
\@onlypreamble\ekvoProcessUnusedGlobalOptions
\@onlypreamble\ekvoProcessOptionsList
\@onlypreamble\ekvoUseUnknownHandlers
```

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.