

exp_kv|OPT

parse class and package options with exp_kv

Jonathan P. Spratte*

2020-07-02 v0.1

Abstract

exp_kv|OPT provides option parsing for classes and packages in $\text{\LaTeX 2}_{\epsilon}$ based on exp_kv. Global and local options are parsed individually by different commands. The stylised name is exp_kv|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

1	Documentation	2
1.1	Macros	2
1.2	Example	3
1.3	Bugs	5
1.4	License	5
2	Implementation	6
2.1	Loop	6
2.2	Tests	6
2.3	Key handlers	7
2.4	Processing list elements	9
2.5	List variable helpers	10
2.6	Errors	11
2.7	User Interface	11

Index	14
--------------	-----------

*jspratte@yahoo.de

1 Documentation

The `expkv` family provides at its core a $\langle key \rangle = \langle value \rangle$ parser and additionally packages, one to conveniently define new keys (`expkvDEF`) and another to build expandable $\langle key \rangle = \langle value \rangle$ taking control sequences (`expkvICS`). Still missing from the mix was a solution to parse \LaTeX 2_ϵ class and package options, a gap that's hereby filled with `expkvOPT`.

`expkvOPT` shouldn't place any restrictions on the keys, but note that parts of \LaTeX 2_ϵ can break if the $\langle key \rangle = \langle value \rangle$ list contains braces. This includes the global options list depending on which class you're using. Also keep in mind that every value provided should be safe from an `\edef` expansion, as the space stripping code of \LaTeX 2_ϵ options (which is applied before `expkvOPT` takes control) uses such an expansion.

The package can be loaded with

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

Unlike the other packages in the `expkv` family, `expkvOPT` 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 `expkvDEF`.

1.1 Macros

`expkvOPT`'s behaviour if it encounters a defined or an undefined $\langle key \rangle$ depends on which list is being parsed and whether the current file is a class or not. Of course in every case a defined $\langle key \rangle$'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 separated in $\langle key \rangle$ and $\langle value \rangle$ but as a whole) is stored within the macro `\CurrentOption`.

<code>\ekvoProcessLocalOptions</code>	<code>\ekvoProcessLocalOptions{\langle set \rangle}</code>
---------------------------------------	--

This parses the options which are directly passed to the current class or package for an `expkv \langle set \rangle`.

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

<code>\ekvoProcessGlobalOptions</code>	<code>\ekvoProcessGlobalOptions{\langle set \rangle}</code>
--	---

In \LaTeX 2_ϵ the options given to `\documentclass` are global options. This macro processes the global options for an `expkv \langle set \rangle`.

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>}

If you want to, instead of parsing all global options, you can parse only those global options which weren't yet used by another package or class.

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*

\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>)

With this macro you can change the action **expkvopt** executes if it encounters an undefined <key> for the next (and only the next) list processing macro. The macro <cs₁> will be called if an undefined <key> without a <value> is encountered and get one argument, being the <key>. Analogous the macro <cs₂> 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>.

\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 L^AT_EX. 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

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

Start the package with the typical L^AT_EX standards.

```
\ekvoVersion Store the packages version and date in two macros.
\ekvoDate    1 \newcommand*\ekvoVersion{0.1}
              2 \newcommand*\ekvoDate{2020-07-02}
```

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

And we report who we are and what we need.

```
3 \ProvidesPackage{expkv-opt}
4 [%
5   \ekvoDate\space v\ekvoVersion\space
6   parse class and package options with expkv%
7 ]
8 \RequirePackage{expkv}
```

2.1 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.

```
9 \protected\long\def\ekvo@CurrentOption@loop#1#2%
10 {%
11   \ekvo@CurrentOption@loop@#2\ekv@mark#1,\ekv@stop,\ekvo@tail
12 }
```

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.

```
13 \long\def\ekvo@CurrentOption@loop@#1#2,%
14 {%
15   \ekv@gobble@from@mark@to@stop#2\ekvo@end@loop\ekv@stop
16   \ekv@ifblank{#2}%
17   {}%
18   {%
19     \edef\CurrentOption{\unexpanded\expandafter{\@gobble#2}}%
20     #1{#2}%
21   }%
22   \ekvo@CurrentOption@loop@#1\ekv@mark
23 }
24 \long\def\ekvo@end@loop#1\ekvo@tail{}
```

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

2.2 Tests

These two macros are just fast branching macros.

```
\ekvo@fi@firstoftwo
\ekvo@fi@gobble
25 \long\def\ekvo@fi@firstoftwo\fi\@secondoftwo#1#2{\fi#1}
26 \long\def\ekvo@fi@gobble\fi\@firstofone#1{\fi}
```

(End definition for \ekvo@fi@firstoftwo and \ekvo@fi@gobble.)

`\ekvo@ifx@TF` 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 $\LaTeX_{2\epsilon}$).

```

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

```

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

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

```

29 \protected\def\ekvo@do@with@set#1#2%
30 {%
31   \ekv@ifdefined{\ekv@undefined@set{#1}}%
32   {%
33     \expandafter
34     \let\expandafter\ekvo@name\csname\ekv@undefined@set{#1}\endcsname
35     \def\ekvo@setname{#1}%
36     #2%
37   }%
38   {\ekvo@err@undefined@set{#1}}%
39 }

```

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

2.3 Key handlers

`\ekv@OPT` uses handlers specifying what happens if a parsed `\key` is defined or undefined.

`\ekvo@handle@undefined@k@pkg` The case for undefined keys in a local list of a package is easy, just throw appropriate
`\ekvo@handle@undefined@kv@pkg` errors.

```

40 \protected\long\def\ekvo@handle@undefined@k@pkg#1%
41 {%
42   \ekv@ifdefined{\ekvo@name{#1}}%
43   {\ekvo@err@value@required{#1}}%
44   {\ekvo@err@undefined@key{#1}}%
45 }
46 \def\ekvo@handle@undefined@kv@pkg#1#2%
47 {%
48   \ekv@ifdefined{\ekvo@name{#1}N}%
49   {\ekvo@err@value@forbidden{#1}}%
50   {\ekvo@err@undefined@key{#1}}%
51 }

```

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

`\ekvo@addto@unused@one` These macros will add or remove the `\CurrentOption` to or from the list of unused global
`\ekvo@addto@unused@two` options.

```

52 \long\def\ekvo@addto@unused@one#1{\ekvo@addto@list\@unusedoptionlist}
53 \long\def\ekvo@addto@unused@two#1#2{\ekvo@addto@list\@unusedoptionlist}
54 \long\def\ekvo@rmfrom@unused@one#1{\ekvo@rmfrom@list\@unusedoptionlist}
55 \long\def\ekvo@rmfrom@unused@two#1#2{\ekvo@rmfrom@list\@unusedoptionlist}

```

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

```

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

```

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

```

56 \protected\def\ekvo@set@handlers@local
57   {%
58     \ekvo@if@need@handlers
59     {%
60       \ifx\@currentx\@clsextension
61       \ifx\@classoptionslist\relax
62         \let\ekvo@handle@undefined@k\@gobble
63         \let\ekvo@handle@undefined@kv\@gobbletwo
64       \else
65         \expandafter
66         \ifx\csname opt@\@currname.\@currentx\endcsname\@classoptionslist
67         \let\ekvo@handle@undefined@k\ekvo@addto@unused@one
68         \let\ekvo@handle@undefined@kv\ekvo@addto@unused@two
69       \else
70         \let\ekvo@handle@undefined@k\@gobble
71         \let\ekvo@handle@undefined@kv\@gobbletwo
72       \fi
73     \fi
74   \else
75     \let\ekvo@handle@undefined@k\ekvo@handle@undefined@k@pkg
76     \let\ekvo@handle@undefined@kv\ekvo@handle@undefined@kv@pkg
77   \fi
78   }%
79 }
80 \protected\def\ekvo@set@handlers@global
81   {%
82     \unless\ifx\@unusedoptionlist\@empty
83       \let\ekvo@handle@defined@k\ekvo@rmfrom@unused@one
84       \let\ekvo@handle@defined@kv\ekvo@rmfrom@unused@two
85     \fi
86     \ekvo@if@need@handlers
87     {%
88       \let\ekvo@handle@undefined@k\@gobble
89       \let\ekvo@handle@undefined@kv\@gobbletwo
90     }%
91   }
92 \protected\def\ekvo@set@handlers@unusedglobal
93   {%
94     \ekvo@if@need@handlers
95     {%
96       \let\ekvo@handle@undefined@k\ekvo@addto@unused@one
97       \let\ekvo@handle@undefined@kv\ekvo@addto@unused@two
98       \let\@unusedoptionlist\@empty
99       \@gobbletwo
100    }%
101    \@firstofone
102    {%
103      \let\ekvo@handle@defined@k\ekvo@rmfrom@unused@one
104      \let\ekvo@handle@defined@kv\ekvo@rmfrom@unused@two
105    }%
106  }
107 \protected\def\ekvo@set@handlers@list

```



```

108   {%
109   \ekvo@if@need@handlers
110   {%
111       \let\ekvo@handle@undefined@k\@gobble
112       \let\ekvo@handle@undefined@kv\@gobbletwo
113   }%
114   }

```

(End definition for \ekvo@set@handlers@local and others.)

`\ekvo@if@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.

```

\ekvo@dont@need@handlers
115 \let\ekvo@if@need@handlers\@firstofone
116 \protected\long\def\ekvo@dont@need@handlers#1%
117   {%
118       \let\ekvo@if@need@handlers\@firstofone
119   }%

```

(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.

```

120 \let\ekvo@handle@defined@k\@gobble
121 \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.

```

122 \protected\def\ekvo@process@common#1#2#3%
123   {%
124       #1{\ekvo@do@with@set{#3}{#2}}%
125   }

```

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

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

```

126 \protected\def\ekvo@process@list#1%
127   {%
128       \expandafter\ekvo@CurrentOption@loop\expandafter{#1}\ekvo@parse
129   }

```

(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}`.

```

130 \long\def\ekvo@parse#1%
131   {%
132       \ekv@eq@other#1\ekv@nil\ekv@mark\ekv@parse@eq@other@a
133       =\ekv@mark\ekv@parse@eq@active\ekv@stop
134       \ekvo@set@k\ekvo@set@kv
135   }

```

(End definition for \ekvo@parse.)

\ekvo@set@k 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.

\ekvo@set@kv

```

136 \protected\def\ekvo@set@k#1%
137   {%
138     \ekv@ifdefined{\ekvo@name{#1}N}%
139     {%
140       \ekvo@handle@defined@k{#1}%
141       \csname\ekvo@name{#1}N\endcsname
142     }%
143     {\ekvo@handle@undefined@k{#1}}%
144   }
145 \protected\def\ekvo@set@kv#1#2%
146   {%
147     \ekv@ifdefined{\ekvo@name{#1}}%
148     {%
149       \ekvo@handle@defined@kv{#1}{#2}%
150       \csname\ekvo@name{#1}\endcsname{#2}%
151     }%
152     {\ekvo@handle@undefined@kv{#1}{#2}}%
153   }

```

(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.

```

154 \protected\def\ekvo@addto@list#1%
155   {%
156     \ekvo@ifx@TF#1\@empty
157     {\let#1\CurrentOption}%
158     {%
159       \edef#1%
160       {%
161         \unexpanded\expandafter{#1},%
162         \unexpanded\expandafter{\CurrentOption}%
163       }%
164     }%
165   }

```

(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 L^AT_EX 2_ε does. We could be faster for items not containing braces, though.

```

166 \protected\def\ekvo@rmfrom@list#1%
167   {%
168     \ekvo@ifx@F#1\@empty
169     {%

```

```

170         \let\ekvo@tmp@list\@empty
171         \let\ekvo@curropt\CurrentOption
172         \expandafter\ekvo@CurrentOption@loop\expandafter{#1}\ekvo@rmfrom@list@
173         \let\CurrentOption\ekvo@curropt
174         \let#1\ekvo@tmp@list
175     }%
176 }
177 \protected\long\def\ekvo@rmfrom@list@#1%
178 {%
179     \ekvo@ifx@F\CurrentOption\ekvo@curropt
180     {\ekvo@addto@list\ekvo@tmp@list}%
181 }

```

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

2.6 Errors

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

```

\ekvo@err@undefined@key \ekvo@err@value@required
\ekvo@err@value@forbidden \ekvo@err@undefined@set
182 \protected\def\ekvo@err@undefined@key#1%
183 {%
184     \PackageError{expkv-opt}{Undefined key ‘#1’ in set ‘\ekvo@setname’}{}%
185 }
186 \protected\def\ekvo@err@value@required#1%
187 {%
188     \PackageError{expkv-opt}%
189     {Value required for key ‘#1’ in set ‘\ekvo@setname’}%
190     {}%
191 }
192 \protected\def\ekvo@err@value@forbidden#1%
193 {%
194     \PackageError{expkv-opt}%
195     {Value forbidden for key ‘#1’ in set ‘\ekvo@setname’}%
196     {}%
197 }
198 \protected\def\ekvo@err@undefined@set#1%
199 {%
200     \PackageError{expkv-opt}%
201     {Undefined set ‘#1’}%
202     {The set for which you try to parse options isn’t defined in expkv.}%
203 }

```

(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

```

204 \protected\def\ekvoProcessLocalOptions
205 {%
206     \ekvo@process@common
207     {\ekv@ifdefined{opt@\@currname.\@currentext}\@firstofone\@gobble}%
208     {%
209         \ekvo@set@handlers@local

```

```

210     \expandafter
211     \ekvo@process@list\csname opt@\@currname.\@currentx\endcsname
212     \AtEndOfPackage{\let\@unprocessedoptions\relax}%
213 }%
214 }

```

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

\ekvoProcessGlobalOptions

```

215 \protected\def\ekvoProcessGlobalOptions
216 {%
217   \ekvo@process@common{\ekvo@ifx@F\@classoptionslist\relax}%
218   {%
219     \ekvo@set@handlers@global
220     \ekvo@process@list\@classoptionslist
221     \let\ekvo@handle@defined@k\@gobble
222     \let\ekvo@handle@defined@kv\@gobbletwo
223   }%
224 }

```

(End definition for \ekvoProcessGlobalOptions. This function is documented on page 2.)

\ekvoProcessUnusedGlobalOptions

```

225 \protected\def\ekvoProcessUnusedGlobalOptions
226 {%
227   \ekvo@process@common{\ekvo@ifx@F\@unusedoptionlist\@empty}%
228   {%
229     \let\ekvo@tmp@list\@unusedoptionlist
230     \ekvo@set@handlers@unusedglobal
231     \ekvo@process@list\ekvo@tmp@list
232     \let\ekvo@handle@defined@k\@gobble
233     \let\ekvo@handle@defined@kv\@gobbletwo
234   }%
235 }

```

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

\ekvoProcessOptionsList

```

236 \protected\def\ekvoProcessOptionsList#1%
237 {%
238   \ekvo@process@common{\ekvo@ifx@F#1\@empty}%
239   {%
240     \ekvo@set@handlers@list
241     \ekvo@process@list#1%
242   }%
243 }

```

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

\ekvoUseUnknownHandlers

```

244 \protected\def\ekvoUseUnknownHandlers#1#2%
245 {%
246   \let\ekvo@handle@undefined@k#1\relax
247   \let\ekvo@handle@undefined@kv#2\relax
248   \let\ekvo@if@need@handlers\ekvo@dont@need@handlers
249 }

```

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

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

```
250 \@onlypreamble\ekvoProcessLocalOptions
251 \@onlypreamble\ekvoProcessGlobalOptions
252 \@onlypreamble\ekvoProcessUnusedGlobalOptions
253 \@onlypreamble\ekvoProcessOptionsList
254 \@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.

C

\CurrentOption 2, 19, 157, 162, 171, 173, 179

E

```

\ekvoDate ..... 1, 3, 5
\ekvoProcessGlobalOptions ... 2, 215, 251
\ekvoProcessLocalOptions ... 2, 204, 250
\ekvoProcessOptionsList ..... 3, 236, 253
\ekvoProcessUnusedGlobalOptions ...
    ..... 3, 225, 252
\ekvoUseUnknownHandlers ..... 3, 244, 254
\ekvoVersion ..... 1, 3, 5

```

T

T_{EX} and L^AT_{EX} 2_ε commands:

```

\ekv@eq@other ..... 132
\ekv@gobble@from@mark@to@stop ... 15
\ekv@ifblank ..... 16
\ekv@ifdefined 31, 42, 48, 138, 147, 207
\ekv@mark ..... 11, 22, 132, 133
\ekv@nil ..... 132
\ekv@parse@eq@active ..... 133
\ekv@parse@eq@other@a ..... 132
\ekv@stop ..... 11, 15, 133
\ekv@undefined@set ..... 31, 34
\ekvo@addto@list .... 52, 53, 154, 180
\ekvo@addto@unused@one .... 52, 67, 96
\ekvo@addto@unused@two .... 52, 68, 97
\ekvo@CurrentOption@loop . 9, 128, 172
\ekvo@CurrentOption@loop@ ..... 9
\ekvo@curropt ..... 171, 173, 179
\ekvo@do@with@set ..... 29, 124
\ekvo@dont@need@handlers .. 115, 248
\ekvo@end@loop ..... 9
\ekvo@err@undefined@key .. 44, 50, 182
\ekvo@err@undefined@set .... 38, 182
\ekvo@err@value@forbidden .. 49, 182

```

```

\ekvo@err@value@required ... 43, 182
\ekvo@fi@firstoftwo ..... 25, 27
\ekvo@fi@gobble ..... 25, 28
\ekvo@handle@defined@k .....
..... 83, 103, 120, 140, 221, 232
\ekvo@handle@defined@kv .....
..... 84, 104, 121, 149, 222, 233
\ekvo@handle@undefined@k .....
.. 62, 67, 70, 75, 88, 96, 111, 143, 246
\ekvo@handle@undefined@k@pkg . 40, 75
\ekvo@handle@undefined@kv .....
.. 63, 68, 71, 76, 89, 97, 112, 152, 247
\ekvo@handle@undefined@kv@pkg 40, 76
\ekvo@if@need@handlers .....
..... 58, 86, 94, 109, 115, 248
\ekvo@ifx@F . 27, 168, 179, 217, 227, 238
\ekvo@ifx@TF ..... 27, 156
\ekvo@name 29, 42, 48, 138, 141, 147, 150
\ekvo@parse ..... 128, 130
\ekvo@process@common .....
..... 122, 206, 217, 227, 238
\ekvo@process@list .....
..... 126, 211, 220, 231, 241
\ekvo@rmfrom@list ..... 54, 55, 166
\ekvo@rmfrom@list@ ..... 166
\ekvo@rmfrom@unused@one .. 52, 83, 103
\ekvo@rmfrom@unused@two .. 52, 84, 104
\ekvo@set@handlers@global .. 56, 219
\ekvo@set@handlers@list .... 56, 240
\ekvo@set@handlers@local ... 56, 209
\ekvo@set@handlers@unusedglobal
..... 56, 230
\ekvo@set@k ..... 134, 136
\ekvo@set@kv ..... 134, 136
\ekvo@setname ..... 29, 184, 189, 195
\ekvo@tail ..... 11, 24
\ekvo@tmp@list . 170, 174, 180, 229, 231

```