

exp_kv|OPT

parse class and package options with exp_kv

Jonathan P. Spratte*

2020-10-10 v0.1b

Abstract

exp_kv|OPT provides option parsing for classes and packages in L^AT_EX_{2 ϵ} 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 exp_kv-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 L^AT_EX 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 L^AT_EX 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 save from an `\edef` expansion, as the space stripping code of L^AT_EX 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 L^AT_EX 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`.

```
\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 L^AT_EX 2_ε 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⟩}`

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 \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

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{0.1b}
          2 \newcommand*\ekvoDate{2020-10-10}
```

(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

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

`\ekvo@ifx@TF` We'll need branching `\ifx` tests so that user input containing unbalanced T_EX ifs doesn't
`\ekvo@ifx@F` break (at least not because of us, everything else is the fault of L^AT_EX 2_ε).

```
25 \def\ekvo@ifx@TF#1#2{\ifx#1#2\ekv@fi@firstoftwo\fi\@secondoftwo}
26 \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` 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.

```

27 \protected\def\ekvo@do@with@set#1#2%
28   {%
29     \ekvifdefinedset{#1}%
30     {%
31       \expandafter
32       \let\expandafter\ekvo@name\csname\ekv@undefined@set{#1}\endcsname
33       \def\ekvo@setname{#1}%
34       #2%
35     }%
36     {\ekvo@err@undefined@set{#1}}%
37   }

```

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

2.3 Key handlers

`\ekvo@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.

```

38 \protected\long\def\ekvo@handle@undefined@k@pkg#1%
39   {%
40     \ekv@ifdefined{\ekvo@name{#1}}%
41     {\ekvo@err@value@required{#1}}%
42     {\ekvo@err@undefined@key{#1}}%
43   }
44 \def\ekvo@handle@undefined@kv@pkg#1#2%
45   {%
46     \ekv@ifdefined{\ekvo@name{#1}N}%
47     {\ekvo@err@value@forbidden{#1}}%
48     {\ekvo@err@undefined@key{#1}}%
49   }

```

(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.

```

50 \long\def\ekvo@addto@unused@one#1{\ekvo@addto@list\@unusedoptionlist}
51 \long\def\ekvo@addto@unused@two#1#2{\ekvo@addto@list\@unusedoptionlist}
52 \long\def\ekvo@rmfrom@unused@one#1{\ekvo@rmfrom@list\@unusedoptionlist}
53 \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` These macros are boring. They just set up the handlers to respect the rules documented
`\ekvo@set@handlers@global` earlier.

```

54 \protected\def\ekvo@set@handlers@local
55   {%
56     \ekvo@if@need@handlers
57     {%

```

```

58     \ifx\@currentx\@clsextension
59     \ifx\@classoptionslist\relax
60     \let\ekvo@handle@undefined@k\@gobble
61     \let\ekvo@handle@undefined@kv\@gobbletwo
62     \else
63     \expandafter
64     \ifx\csname opt@\@currname.\@currentx\endcsname\@classoptionslist
65     \let\ekvo@handle@undefined@k\ekvo@addto@unused@one
66     \let\ekvo@handle@undefined@kv\ekvo@addto@unused@two
67     \else
68     \let\ekvo@handle@undefined@k\@gobble
69     \let\ekvo@handle@undefined@kv\@gobbletwo
70     \fi
71     \fi
72     \else
73     \let\ekvo@handle@undefined@k\ekvo@handle@undefined@k@pkg
74     \let\ekvo@handle@undefined@kv\ekvo@handle@undefined@kv@pkg
75     \fi
76     }%
77 }
78 \protected\def\ekvo@set@handlers@global
79 {%
80     \unless\ifx\@unusedoptionlist\@empty
81     \let\ekvo@handle@defined@k\ekvo@rmfrom@unused@one
82     \let\ekvo@handle@defined@kv\ekvo@rmfrom@unused@two
83     \fi
84     \ekvo@if@need@handlers
85     {%
86     \let\ekvo@handle@undefined@k\@gobble
87     \let\ekvo@handle@undefined@kv\@gobbletwo
88     }%
89 }
90 \protected\def\ekvo@set@handlers@unusedglobal
91 {%
92     \ekvo@if@need@handlers
93     {%
94     \let\ekvo@handle@undefined@k\ekvo@addto@unused@one
95     \let\ekvo@handle@undefined@kv\ekvo@addto@unused@two
96     \let\@unusedoptionlist\@empty
97     \@gobbletwo
98     }%
99     \@firstofone
100    {%
101    \let\ekvo@handle@defined@k\ekvo@rmfrom@unused@one
102    \let\ekvo@handle@defined@kv\ekvo@rmfrom@unused@two
103    }%
104 }
105 \protected\def\ekvo@set@handlers@list
106 {%
107     \ekvo@if@need@handlers
108     {%
109     \let\ekvo@handle@undefined@k\@gobble
110     \let\ekvo@handle@undefined@kv\@gobbletwo
111     }%

```



```
112 }
```

(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
`\ekvo@dont@need@handlers` will act like `\@gobble` and also let it to `\@firstofone` afterwards.

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

(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.

```
118 \let\ekvo@handle@defined@k\@gobble
119 \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.

```
120 \protected\def\ekvo@process@common#1#2#3%
121   {%
122     #1{\ekvo@do@with@set{#3}{#2}}%
123   }
```

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

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

```
124 \protected\def\ekvo@process@list#1%
125   {%
126     \expandafter\ekvo@CurrentOption@loop\expandafter{#1}\ekvo@parse
127   }
```

(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}`.

```
128 \protected\long\def\ekvo@parse#1%
129   {%
130     \ekv@eq@other#1\ekv@nil\ekv@mark\ekv@parse@eq@other@a
131     =\ekv@mark\ekv@parse@eq@active
132     \ekvo@set@k\ekvo@set@kv
133     \ekvo@tail
134   }
```

(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`.

```

135 \protected\def\ekvo@set@k#1#2\ekvo@tail
136   {%
137     \ekv@ifdefined{\ekvo@name{#1}N}%
138     {%
139       \ekvo@handle@defined@k{#1}%
140       \csname\ekvo@name{#1}N\endcsname
141     }%
142     {\ekvo@handle@undefined@k{#1}}%
143   }
144 \protected\def\ekvo@set@kv#1#2#3\ekvo@tail
145   {%
146     \ekv@ifdefined{\ekvo@name{#1}}%
147     {%
148       \ekvo@handle@defined@kv{#1}{#2}%
149       \csname\ekvo@name{#1}\endcsname{#2}%
150     }%
151     {\ekvo@handle@undefined@kv{#1}{#2}}%
152   }

```

(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.

```

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

```

(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 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}_{2_{\epsilon}}$ does. We could be faster for items not containing braces, though.

```

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

```

```

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

```

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

```

(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

```

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

```

```

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

```

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

`\ekvoProcessGlobalOptions`

```

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

```

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

`\ekvoProcessUnusedGlobalOptions`

```

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

```

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

`\ekvoProcessOptionsList`

```

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

```

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

`\ekvoUseUnknownHandlers`

```

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

```

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

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

```
249 \@onlypreamble\ekvoProcessLocalOptions
250 \@onlypreamble\ekvoProcessGlobalOptions
251 \@onlypreamble\ekvoProcessUnusedGlobalOptions
252 \@onlypreamble\ekvoProcessOptionsList
253 \@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	
<code>\CurrentOption</code>	<i>2, 19, 156, 161, 170, 172, 178</i>
E	
<code>\ekvifdefinedset</code>	<i>29</i>
<code>\ekvoDate</code>	<i><u>1</u>, 3, 5</i>
<code>\ekvoProcessGlobalOptions</code>	<i>2, <u>214</u>, 250</i>
<code>\ekvoProcessLocalOptions</code>	<i>2, 203, 249</i>
<code>\ekvoProcessOptionsList</code>	<i>3, <u>235</u>, 252</i>
<code>\ekvoProcessUnusedGlobalOptions</code>	<i>3, <u>224</u>, 251</i>
<code>\ekvoUseUnknownHandlers</code>	<i>3, <u>243</u>, 253</i>
<code>\ekvoVersion</code>	<i><u>1</u>, 3, 5</i>
T	
T _E X and L ^A T _E X 2 _ε commands:	
<code>\ekv@eq@other</code>	<i>130</i>
<code>\ekv@fi@firstoftwo</code>	<i>25</i>
<code>\ekv@fi@gobble</code>	<i>26</i>
<code>\ekv@gobble@from@mark@to@stop</code>	<i>15</i>
<code>\ekv@ifblank</code>	<i>16</i>
<code>\ekv@ifdefined</code>	<i>40, 46, 137, 146, 206</i>
<code>\ekv@mark</code>	<i>11, 22, 130, 131</i>
<code>\ekv@nil</code>	<i>130</i>
<code>\ekv@parse@eq@active</code>	<i>131</i>
<code>\ekv@parse@eq@other@a</code>	<i>130</i>
<code>\ekv@stop</code>	<i>11, 15</i>
<code>\ekv@undefined@set</code>	<i>32</i>
<code>\ekvo@addto@list</code>	<i>50, 51, <u>153</u>, 179</i>
<code>\ekvo@addto@unused@one</code>	<i>50, 65, 94</i>
<code>\ekvo@addto@unused@two</code>	<i>50, 66, 95</i>
<code>\ekvo@CurrentOption@loop</code>	<i>9, <u>126</u>, 171</i>
<code>\ekvo@CurrentOption@loop@</code>	<i>9</i>
<code>\ekvo@curropt</code>	<i>170, 172, 178</i>
<code>\ekvo@do@with@set</code>	<i>27, 122</i>
<code>\ekvo@dont@need@handlers</code>	<i>113, 247</i>
<code>\ekvo@end@loop</code>	<i><u>9</u></i>
<code>\ekvo@err@undefined@key</code>	<i>42, 48, <u>181</u></i>
<code>\ekvo@err@undefined@set</code>	<i>36, <u>181</u></i>
<code>\ekvo@err@value@forbidden</code>	<i>47, <u>181</u></i>
<code>\ekvo@err@value@required</code>	<i>41, <u>181</u></i>
<code>\ekvo@handle@defined@k</code>	<i>81, 101, 118, 139, 220, 231</i>
<code>\ekvo@handle@defined@kv</code>	<i>82, 102, 119, 148, 221, 232</i>
<code>\ekvo@handle@undefined@k</code>	<i>60, 65, 68, 73, 86, 94, 109, 142, 245</i>
<code>\ekvo@handle@undefined@k@pkg</code>	<i>38, 73</i>
<code>\ekvo@handle@undefined@kv</code>	<i>61, 66, 69, 74, 87, 95, 110, 151, 246</i>
<code>\ekvo@handle@undefined@kv@pkg</code>	<i>38, 74</i>
<code>\ekvo@if@need@handlers</code>	<i>56, 84, 92, 107, <u>113</u>, 247</i>
<code>\ekvo@ifx@F</code>	<i>25, 167, 178, 216, 226, 237</i>
<code>\ekvo@ifx@TF</code>	<i>25, 155</i>
<code>\ekvo@name</code>	<i>27, 40, 46, 137, 140, 146, 149</i>
<code>\ekvo@parse</code>	<i>126, <u>128</u></i>
<code>\ekvo@process@common</code>	<i>120, 205, 216, 226, 237</i>
<code>\ekvo@process@list</code>	<i>124, 210, 219, 230, 240</i>
<code>\ekvo@rmfrom@list</code>	<i>52, 53, <u>165</u></i>
<code>\ekvo@rmfrom@list@</code>	<i>165</i>
<code>\ekvo@rmfrom@unused@one</code>	<i>50, 81, 101</i>
<code>\ekvo@rmfrom@unused@two</code>	<i>50, 82, 102</i>
<code>\ekvo@set@handlers@global</code>	<i>54, 218</i>
<code>\ekvo@set@handlers@list</code>	<i>54, 239</i>
<code>\ekvo@set@handlers@local</code>	<i>54, 208</i>
<code>\ekvo@set@handlers@unused@global</code>	<i>54, 229</i>
<code>\ekvo@set@k</code>	<i>132, <u>135</u></i>
<code>\ekvo@set@kv</code>	<i>132, <u>135</u></i>
<code>\ekvo@setname</code>	<i>27, 183, 188, 194</i>
<code>\ekvo@tail</code>	<i>11, 24, 133, 135, 144</i>
<code>\ekvo@tmp@list</code>	<i>169, 173, 179, 228, 230</i>