The kvoptions package

Heiko Oberdiek
<oberdiek@uni-freiburg.de>

2006/08/22 v2.4

Abstract

This package is intended for package authors who want to use options in key value format for their package options.

Contents

1 Introduction

1.1 The beginning

This package addresses class or package writers that want to allow their users to specify options as key value pairs, e.g.:

```
\documentclass[verbose=false,name=me]{myclass}
\usepackage[format=print]{mylayout}
```

Prominent example is package hyperref, probably the first package that offers this service. It's `\ProcessOptionsWithKV` is often copied und used in other packages, e.g. package helvet that uses this interface for its option scaled.

However copying code is not the most modern software development technique. And hyperref's code for `\ProcessOptionsWithKV` was changed to fix bugs. The version used in other packages depends on the time of copying and the awareness of hyperref's changes. Now the code is sourced out into this package and available for other package or class writers.

1.2 Overview

Package kvoptions connects package *keyval* with LaTeX's package and class *options*:

Package keyval	**Package kvoptions**	**LaTeX kernel**

`\define@key`	`\DeclareVoidOption`
	`\DeclareStringOption`
	`\DeclareBoolOption`
	`\DeclareComplementaryOption`
	`\DisableKeyvalOption`
		`\DeclareOption`

	`\DeclareDefaultOption`	`\DeclareOption*`
	`\ProcessKeyvalOptions`	`\ProcessOptions*`
	Option patch	Class/package option system
	`\SetupKeyvalOptions`	

2 Usage

2.1 Process options

2.1.1 `\ProcessKeyvalOptions`

```
\ProcessKeyvalOptions {⟨family⟩}
\ProcessKeyvalOptions *
```

This command evaluates the global or local options of the package that are defined with keyval's interface within the family ⟨*family*⟩. It acts the same way as LaTeX's `\ProcessOptions*`. In a package unknown global options are ignored, in a class they are added to the unknown option list. The known global options and all local options are passed to keyval's `\setkeys` command for executing the options. Unknown options are reported to the user by an error.

If the family name happens to be the same as the name of the package or class where `\ProcessKeyvalOptions` is used or the family name has previously been setup by `\SetupKeyvalOptions`, then `\ProcessKeyvalOptions` knows the family name already and you can use the star form without mandatory argument.

Note, neither of the following commands are necessary for `\ProcessKeyvalOptions`. They just help the package/class author in common tasks.

2.1.2 `\SetupKeyvalOptions`

```
\SetupKeyvalOptions {
   family = ⟨family⟩ ,
   prefix = ⟨prefix⟩
}
```

This command allows to configure the default assumptions that are based on the current package or class name. LaTeX remembers this name in `\@currname`. The syntax description of the default looks a little weird, therefor an example is given for a package or class named `foobar`.

Key	Default	(example)	Used by

`family`	⟨`\currname`⟩	(`foobar`)	`\ProcessKeyvalOptions*`
			`\DeclareBoolOption`
			`\DeclareStringOption`

`prefix`	⟨`\currname`⟩`@`	(`foobar@`)	`\DeclareBoolOption`
			`\DeclareStringOption`
			`\DeclareVoidOption`

2.2 Option declarations

The options for `\ProcessKeyvalOptions` are defined by keyval's `\define@key`. Common purposes of such keys are boolean switches, they enable or disable something. Or they store a name or some kind of string in a macro. The following commands help the user. He declares what he wants and kvoptions take care of the key definition, resource allocation and initialization.

In order to avoid name clashes of macro names, internal commands are prefixed. Both the prefix and the family name for the defined keys can be configured by `\SetupKeyvalOptions`.

2.2.1 `\DeclareStringOption`

`\DeclareStringOption` [⟨*init*⟩] {⟨*key*⟩} [⟨*default*⟩]

A macro is created that remembers the value of the key ⟨*key*⟩. The name of the macro consists of the option name ⟨*key*⟩ that is prefixed by the prefix (see 2.1.2). The initial contents of the macro can be given by the first optional argument ⟨*init*⟩. The default is empty.

The the option ⟨*key*⟩ is defined. The option code just stores its value in the macro. If the optional argument at the end of `\DeclareStringOption` is given, then option ⟨*key*⟩ is defined with the default ⟨*default*⟩.

Example for a package with the following two lines:

```
\ProvidedPackage{foobar}
\DeclareStringOption[me]{name}
```

Then `\DeclareStringOption` defines the macro with content `me`, note LaTeX complains if the name of the macro already exists:

```
\newcommand*{\foobar@name}{me}
```

The option definition is similar to:

```
\define@key{foobar}{name}{%
  \renewcommand*{\foobar@name}{#1}%
}
```

2.2.2 `\DeclareBoolOption`

`\DeclareBoolOption` [⟨*init*⟩] {⟨*key*⟩}

A boolean switch is generated, initialized by value ⟨*init*⟩ and the corresponding key ⟨*key*⟩ is defined. If the initialization value is not given, `false` is used as default.

The internal actions of `\DeclareBoolOption` are shown below. The example is given for a package author who has the following two lines in his package/class:

```
\ProvidesPackage{foobar}
\DeclareBoolOption{verbose}
```

First a new switch is created:

```
\newif\iffoobar@verbose
```

and initialized:

```
\foobar@verbosefalse
```

Finally the key is defined:

```
\define@key{foobar}{verbose}[true]{...}
```

The option code configures the boolean option in the following way: If the author specifies `true` or `false` then the switch is turned on or off respectivly. Also the option can be given without explicit value. Then the switch is enabled. Other values are reported as errors.

Now the switch is ready to use in the package/class, e.g.:

```
\iffoobar@verbose
% print verbose message
\else
% be quiet
\fi
```

Users of package `\ifthen` can use the switch as boolean:

```
\boolean{foobar@verbose}
```

2.2.3 `\DeclareComplementaryOption`

`\DeclareComplementaryOption` {⟨*key*⟩} {⟨*parent*⟩}

Sometimes contrasting names are used to characterize the two states of a boolean switch, for example final vs. final. Both options behave like boolean options but they do not need to different switches, they should share one. `\DeclareComplementaryOption` allows this. The option ⟨*key*⟩ shares the switch of option ⟨*parent*⟩. Example:

```
\DeclareBoolOption{draft}
\DeclareComplementaryOption{final}{draft}
```

Then `final` sets the switch of draft to `false`, and `final=false` enables the draft switch.

2.2.4 `\DeclareVoidOption`

`\DeclareVoidOption` {⟨*key*⟩} {⟨*code*⟩}

`\ProcessKeyvalOptions` can be extended to recognize options that are declared in traditional way by `\DeclareOption`. But in case of the error that the user specifies a value, then this option would not recognized as key value option because of `\DeclareOption` and not detected as traditional option because of the value part. The user would get an unknown option error, difficult to understand.

`\DeclareVoidOption` solves this problem. It defines the option ⟨*key*⟩ as key value option. If the user specifies a value, a warning is given and the value is ignored.

The code part ⟨*code*⟩ is stored in a macro. The name of the macro consists of the option name ⟨*key*⟩ that is prefixed by the prefix (see 2.1.2). If the option is set, the macro will be executed. During the execution `\CurrentOption` is available with the current key name.

2.2.5 `\DeclareDefaultOption`

`\DeclareDefaultOption` {⟨*code*⟩}

This command does not define a key, it is the equivalent to LaTeX's `\DeclareOption*`. It allows the specification of a default action ⟨*code*⟩ that is invoked if an unknown option is found. During the execution of ⟨*code*⟩ `\CurrentOption` contains the current option string. Additionally `\CurrentOptionValue` contains the value part if the option string is parsable as key value pair, otherwise it is `\relax`. `\CurrentOptionKey` contains the key of the key value pair, or the whole option string, if it misses the equal sign.

Inside packages typical default actions are to pass unknown options to another package. Or an error message can be thrown by `\@unknownoptionerror`. This is the original error message that LaTeX gives for unkown package options. This error message is easier to understand for the user as the error message from package keyval that is given otherwise.

A Class ignores unknown options and puts them on the unused option list. Let LaTeX do the job and just call `\OptionNotUsed`. Or the options can be passed to another class that is later loaded.

2.2.6 Dynamic options

Options of LaTeX's package/class system are cleared in `\ProcessOptions`. They modify the static model of a package. For example, depending on option bookmarks package hyperref loads differently.

Options, however, defined by keyval's `\define@key` remain defined, if the options are processed by `\setkeys`. Therefore these options can also be used to model the dynamic behaviour of a package. For example, in hyperref the link colors can be changed everywhere until the end in `\end{document}`.

However package color is necessary and it cannot be loaded after `\begin{document}`. Option colorlinks that loads color should be active until `\begin{document}` and die in some way if it is too late for loading packages. With `\DisableKeyvalOption` the package/class author can specify and configure the death of an option and controls the life period of the option.

2.2.7 `\DisableKeyvalOption`

`\DisableKeyvalOption` [⟨*options*⟩] {⟨*family*⟩} {⟨*key*⟩}

⟨*options*⟩:

`action`	= `undef`, `warning`, `error`, or `ignore`	default: `undef`
`global` or `local`		default: `global`
`package` or `class` = ⟨*name*⟩		

`\DisableKeyvalOption` can be called to mark the end when the option ⟨*key*⟩ is no longer useful. The behaviour of an option after its death can be configured by action:

`undef`: The option will be undefined, If it is called, `\setkeys` reports an error because of unknown key.

`error` or `warning`: Use of the option will cause an error or warning message. Also these actions require that exclusivly either the package or class name is given in options package or class.

`ignore`: The use of the option will silently be ignored.

The option's death can be limited to the end of the current group, if option local is given. Default is global.

The package/class author can wish the end of the option already during the package loading, then he will have static behaviour. In case of dynamic options `\DisableKeyvalOptions` can be executed everywhere, also outside the package. Therefore the family name and the package/class name is usually unknown for `\DisableKeyvalOptions`. Therefore the argument for the family name is mandatory and for some actions the package/class name must be provided.

Usually a macro would configure the option death, Example:

```
\ProvidesPackage{foobar}
\DeclareBoolOption{color}
\DeclareStringOption[red]{emphcolor}
\ProcessKeyvalOptions*

\newcommand*{\foobar@DisableOption}[2]{%
  \DisableKeyvalueOption[
    action={#1},
    package=foobar
  ]{foobar}{#2}%
}

\iffoobar@color
  \RequirePackage{color}
  \renewcommand*{\emph}[1]{\textcolor{\foobar@emphcolor}{#1}}
\else
  % Option emphcolor is not wrong, if we have color support.
  % otherwise the option has no effect, but we don't want to
  % remove it. Therefore action 'ignore' is the best choice:
```

```
  \foobar@DisableOption{ignore}{emphcolor}
\fi
% No we don't need the option 'color'.
\foobar@DisableOption{warning}{color}

% With color support option 'emphcolor' will dynamically change
% the color of \emph statements.
```

2.3 Summary of internal macros

The `\Declare...Option` commands define macros, additionally to the macros generated by the key definition. These macros can be used by the package/class author. The name of the macros starts with the prefix ⟨*prefix*⟩ that can be configured by `\SetupKeyvalOptions`.

Declare ⟨*key*⟩	Defined macro	Description
`\DeclareStringOption`	`\`⟨*prefix*⟩⟨*key*⟩	holds the string
`\DeclareBoolOption`	`\if`⟨*prefix*⟩⟨*key*⟩	boolean switch
	`\`⟨*prefix*⟩⟨*key*⟩`false`	disable switch
	`\`⟨*prefix*⟩⟨*key*⟩`true`	enable switch
`\DeclareComplementaryOption`	`\`⟨*prefix*⟩⟨*key*⟩`false`	enable parent switch
	`\`⟨*prefix*⟩⟨*key*⟩`true`	disable parent switch
`\DeclareVoidOption`	`\`⟨*prefix*⟩⟨*key*⟩	holds the action

3 Example

The following example defined a package that serves some private color management. A boolean option print enables print mode without colors. An option emph redefines `\emph` to print in the given color. And the driver can be specified by option driver.

```
⟨*example⟩
    % Package identification
    % ----------------------
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{example-mycolorsetup}[2006/08/22 Managing my colors]

\RequirePackage{ifpdf}
\RequirePackage{kvoptions}

    % Option declarations
    % -------------------

\SetupKeyvalOptions{
  family=MCS,
  prefix=MCS@
}
    % Use a shorter family name and prefix

    % Option print
\DeclareBoolOption{print}
    % is the same as
    % \DeclareBoolOption[false]{print}

    % Option driver
\ifpdf
  \DeclareStringOption[pdftex]{driver}
\else
  \DeclareStringOption[dvips]{driver}
\fi

```

```
    % Alternative interface for driver options
\DeclareVoidOption{dvips}{\SetupDriver}
\DeclareVoidOption{dvipdfm}{\SetupDriver}
\DeclareVoidOption{pdftex}{\SetupDriver}
    % In \SetupDriver we take the current option \CurrentOption
    % and pass it to the driver option.
    % The \expandafter commands expand \CurrentOption at the
    % time, when \SetupDriver is executed and \CurrentOption
    % has the correct meaning.
\newcommand*{\SetupDriver}{%
  \expandafter\@SetupDriver\expandafter{\CurrentOption}%
}
\newcommand*{\@SetupDriver}[1]{%
  \setkeys{MCS}{driver={#1}}%
}

    % Option emph
    % An empty value means, we want to have no color for \emph.
    % If the user specifies option emph without value, the red is used.
\DeclareStringOption{emph}[red]
    % is the same as
    % \DeclareStringOption[]{emph}[red]

    % Default option rule
\DeclareDefaultOption{%
  \ifx\CurrentOptionValue\relax
    \PackageWarningNoLine{\@currname}{%
      Unknown option ‘\CurrentOption’\MessageBreak
      is passed to package ‘color’%
    }%
    % Pass the option to package color.
    % Again it is better to expand \CurrentOption.
    \expandafter\PassOptionsToPackage\expandafter{\CurrentOption}{color}%
  \else
    % Package color does not take options with values.
    % We provide the standard LaTeX error.
    \@unknownoptionerror
  \fi
}

    % Process options
    % ---------------
\ProcessKeyvalOptions*

    % Implementation depending on option values
    % ----------------------------------------
    % Code for print mode
\ifMCS@print
  \PassOptionsToPackage{monochrome}{color}
    % tells package color to use black and white
\fi

\RequirePackage[\MCS@driver]{color}
    % load package color with the correct driver

    % \emph setup
\ifx\MCS@emph\@empty
    % \@empty is a predefined macro with empty contents.
    % the option value of option emph is empty, thus
    % we do not want a redefinition of \emph.
\else
  \renewcommand*{\emph}[1]{%
```

```
    \textcolor{\MCS@emph}{#1}%
  }
\fi
⟨/example⟩
```

4 Package options

The package kvoptions knows two package options patch and debugshow. The options of package kvoptions are intended for authors, not for package/class writers. Inside a package it is too late for option patch and debugshow enables some messages that are perhaps useful for the debugging phase. Also LaTeX is unhappy if a package is loaded later again with options that are previously not given. Thus package and class authors, stay with `\RequirePackage{kvoptions}` without options.

4.1 Option patch

LaTeX's system of package/class options has some severe limitations that especially affects the value part if options are used as pair of key and value.

- Spaces are removed, regardless where:

  ```
  \documentclass[box=0 0 400 600]{article}
  ```

  Now each package will see box=00400600 as global option.

- In the previous case also braces would not help:

  ```
  \documentclass[box={0 0 400 600}]{article}
  ```

  The result is an error message:

  ```
  ! LaTeX Error: Missing \begin{document}.
  ```

  As local option, however, it works if the package knows about key value options (By using this package, for example).

- The requirements on robustness are extremly high. LaTeX expands the option. All that will not work as environment name will break also as option. Even a `\relax` will generate an error message:

  ```
  ! Missing \endcsname inserted.
  ```

  Of course, LaTeX does not use its protecting mechanisms. On contrary `\protect` itself will cause errors.

- The options are expanded. But perhaps the package will do that, because it has to setup some things before? Example hyperref:

  ```
  \usepackage[pdfauthor=M\"uller]{hyperref}
  ```

  Package hyperref does not see `M\"uller` but its expansion and it does not like it, you get many warnings

  ```
  Token not allowed in a PDFDocEncoded string
  ```

  And the title becomes: `Mu127uller`. Therefore such options must usually be given after package hyperref is loaded:

  ```
  \usepackage{hyperref}
  \hypersetup{pdfauthor=Fran\c coise M\"uller}
  ```

  As package option it will even break with `Fran\c coise` because of the cedilla `\c c`, it is not robust enough.

For users that do not want with this limitations the package offers option patch. It patches LaTeX's option system and tries to teach it also to handle options that are given as pairs of key and value and to prevent expansion. It can already be used at the very beginning, before `\documentclass`:

```
\RequirePackage[patch]{kvoptions}
\documentclass[pdfauthor=Fran\c coise M\"uller]{article}
\usepackage{hyperref}
```

The latest time is before the package where you want to use problematic values:

```
\usepackage[patch]{kvoptions}
\usepackage[Fran\c coise M\"uller]{hyperref}
```

Some remarks:

- The patch and requires ε-TeX, its `\unexpanded` feature is much to nice. It is possible to work around using token registers. But the code becomes longer, slower, more difficult to read and maintain. The package without option patch works and will work without ε-TeX.
- The code for the patch is quite long, there are many test cases. Thus the probability for bugs is probably not too small.

4.2 Option debugshow

The name of this option follows the convention of packages multicol, tabularx, and tracefnt. Currently it prints the setting of boolean options, declared by `\DeclareBoolOption` in the `.log` file, if that boolean option is used. You can activate the option by

- `\PassOptionsToPackage{debugshow}{kvoptions}`
  Put this somewhere before package kvoptions is loaded first, e.g. before `\documentclass`.
- `\RequirePackage[debugshow]{kvoptions}`
  Before `\documentclass` even an author has to use `\RequirePackage`. `\usepackage` only works after `\documentclass`.

The preferred method is `\PassOptionsToPackage`, because it does not force the package loading and does not disturb, if the package is not loaded later at all.

5 Limitations

5.1 Compatibility

5.1.1 Option patch vs. package xkvltxp

Package xkvltxp from the xkeyval project has the same goal as kvoptions's option patch and patches LaTeX's kernel commands. Of course they cannot be used both. The user must decide, which method he prefers. Package kvoptions drops option patch if it detects that xkvltxp is already loaded.

However both *packages* can be used together, example:

```
\usepackage{xkvltxp}
\usepackage[...]{foobar} % foobar using kvoptions
```

The other way should work, too.

Package kvoptions tries to catch more situations and to be more robust. For example, during the comparison of options it normalizes them by removing spaces around = and the value. Thus the following is not reported as option clash:

```
\RequirePackage[patch]{kvoptions}
\documentclass{article}

\usepackage[scaled=0.7]{helvet}
\usepackage[scaled = 0.7]{helvet}

\begin{document}
\end{document}
```

5.2 Limitations

5.2.1 Option comparisons

In some situations LaTeX compares options, e.g. option clash check, `\@ifpackagewith`, `\@ifclasswith`. Apart from catcode and sanitizing problems of option `patch`, there is another problem. LaTeX does not know about the type and default values of options in key value style. Thus an option clash is reported, even if the key value has the same meaning:

```
\usepackage[scaled]{helvet} % default is .95
\usepackage[.95]{helvet}
\usepackage[0.95]{helvet}
```

5.2.2 Option list parsing with option patch

With option `patch` the range of possible values in key value specifications is much large, for example the comma can be used, if enclosed in curly braces.

Other packages, especially the packages that uses their own process option code can be surprised to find tokens inside options that they do not expect and errors would be the consequence. To avoid errors the options, especially the unused option list is sanitized. That means the list will only contain tokens with catcode 12 (other) and perhaps spaces (catcode 10). This allows a safe parsing for other packages. But a comma in the value part is no longer protected by curly braces because they have lost their special meaning. This is the price for compatibility.

Example:

```
\RequirePackage[patch]{kvoptions}
\documentclass[a={a,b,c},b]{article}
\begin{document}
\end{document}
```

Result:

```
LaTeX Warning: Unused global option(s):
    [a={a,c},b].
```

6 Implementation

6.1 Preamble

Package identification.

```
⟨*package⟩
\NeedsTeXFormat{LaTeX2e}
\ProvidesPackage{kvoptions}
  [2006/08/22 v2.4 Connects package keyval with LaTeX options (HO)]
```

External resources. The package extends the support for key value pairs of package `\keyval` to package options. Thus the package needs to be loaded anyway. and we use it for `\SetupKeyvalOptions`. AFAIK this does not disturb users of xkeyval.

```
\RequirePackage{keyval}
```

Options Option debugshow enables additional lines of code that prints information into the `.log` file.

```
\begingroup
  \edef\x{\endgroup
    \noexpand\AtEndOfPackage{%
      \catcode\@ne=\the\catcode\@ne\relax
      \catcode\tw@=\the\catcode\tw@\relax
    }%
  }%
\x
\catcode\@ne=14 %
\catcode\tw@=14 %
\DeclareOption{debugshow}{\catcode\@ne=9 }
```

```
\DeclareOption{patch}{\catcode\tw@=9 }
```

Optionen auswerten:

```
\ProcessOptions\relax
```

6.2 Option declaration macros

6.2.1 `\SetupKeyvalOptions`

The family for the key value pairs can be setup once and is remembered later. The package name seems a reasonable default for the family key, if it is not set by the package author.

`\KVO@family` We cannot store the family setting in one macro, because the package should be usable for many other packages, too. Thus we remember the family setting in a macro, whose name contains the package name with extension, a key in LaTeX's class/package system.

```
\define@key{KVO}{family}{%
  \expandafter\edef\csname KVO@family@\@currname.\@currext\endcsname{#1}%
}
\def\KVO@family{%
  \@ifundefined{KVO@family@\@currname.\@currext}{%
    \@currname
  }{%
    \csname KVO@family@\@currname.\@currext\endcsname
  }%
}
```

`\KVO@prefix` The value settings of options, declared by `\DeclareBoolOption` and `\DeclareStringOption` need to be saved in macros. in the first case this is a switch `\if`⟨*prefix*⟩⟨*key*⟩, in the latter case a macro `\`⟨*prefix*⟩⟨*key*⟩. The prefix can be configured, by prefix that is declared here. The default is the package name with `@` appended.

```
\define@key{KVO}{prefix}{%
  \expandafter\edef\csname KVO@prefix@\@currname.\@currext\endcsname{#1}%
}
\def\KVO@prefix{%
  \@ifundefined{KVO@prefix@\@currname.\@currext}{%
    \@currname @%
  }{%
    \csname KVO@prefix@\@currname.\@currext\endcsname
  }%
}
```

`\SetupKeyvalOptions` The argument of `\SetupKeyvalOptions` expects a key value list, known keys are family and prefix.

```
\newcommand*{\SetupKeyvalOptions}{%
  \setkeys{KVO}%
}
```

6.2.2 `\DeclareBoolOption`

`\DeclareBoolOption` Usually options of boolean type can be given by the user without value and this means a setting to *true*. We follow this convention here. Also it simplifies the user interface.

The switch is created and initialized with *false*. The default setting can be overwritten by the optional argument.

LaTeX's `\newif` does not check for already defined macros, therefore we add this check here to prevent the user from accidently redefining of TeX's primitives and other macros.

```
\newcommand*{\DeclareBoolOption}[2][false]{%
  \KVO@ifdefinable{if\KVO@prefix#2}{%
    \KVO@ifdefinable{\KVO@prefix#2true}{%
      \KVO@ifdefinable{\KVO@prefix#2false}{%
        \expandafter\newif\csname if\KVO@prefix#2\endcsname
        \@ifundefined{\KVO@prefix#2#1}{%
          \PackageWarning{kvoptions}{%
            Initialization of option ‘#2’ failed,\MessageBreak
            cannot set boolean option to ‘#1’,\MessageBreak
            use ‘true’ or ‘false’, now using ‘false’%
          }%
        }{%
          \csname\KVO@prefix#2#1\endcsname
        }%
        \begingroup
          \edef\x{\endgroup
            \noexpand\define@key{\KVO@family}{#2}[true]{%
              \noexpand\KVO@boolkey{\@currname}%
              \ifx\@currext\@clsextension
                \noexpand\@clsextension
              \else
                \noexpand\@pkgextension
              \fi
              {\KVO@prefix}{#2}{####1}%
            }%
          }%
        \x
      }%
    }%
  }%
}
```

`\DeclareComplementaryOption` The first argument is the key name, the second the key that must be a boolean option with the same current family and prefix. A new switch is not created for the new key, we have already a switch. Instead we define switch setting commands to work on the parent switch.

```
\newcommand*{\DeclareComplementaryOption}[2]{%
  \@ifundefined{if\KVO@prefix#2}{%
    \PackageError{kvoptions}{%
      Cannot generate option code for ‘#1’,\MessageBreak
      parent switch ‘#2’ does not exist%
    }{%
      You are inside %
      \ifx\@currext\@clsextension class\else package\fi\space
      ‘\@currname.\@currext’.\MessageBreak
      `\KVO@family' is used as familiy for the keyval options.\MessageBreak
      `\KVO@prefix' serves as prefix for internal switch macros.\MessageBreak
      \MessageBreak
      \@ehc
    }%
  }{%
    \KVO@ifdefinable{\KVO@prefix#1true}{%
      \KVO@ifdefinable{\KVO@prefix#1false}{%
        \expandafter\let\csname\KVO@prefix#1false\expandafter\endcsname
          \csname\KVO@prefix#2true\endcsname
        \expandafter\let\csname\KVO@prefix#1true\expandafter\endcsname
          \csname\KVO@prefix#2false\endcsname
```

The same code part as in `\DeclareBoolOption` can now be used.

```
        \begingroup
          \edef\x{\endgroup
            \noexpand\define@key{\KVO@family}{#1}[true]{%
              \noexpand\KVO@boolkey{\@currname}%
              \ifx\@currext\@clsextension
                \noexpand\@clsextension
              \else
                \noexpand\@pkgextension
              \fi
              {\KVO@prefix}{#1}{####1}%
            }%
          }%
        \x
      }%
    }%
  }%
}
```

`\KVO@ifdefinable` Generate the command token LaTeX's `\@ifdefinable` expects.

```
\def\KVO@ifdefinable#1{%
  \expandafter\@ifdefinable\csname #1\endcsname
}
```

`\KVO@boolkey` We check explicitly for `true` and `false` to prevent the user from accidently calling other macros.

`#1`	package/class name
`#2`	`\@pkgextension/\@clsextension`
`#3`	prefix
`#4`	key name
`#5`	new value

```
\def\KVO@boolkey#1#2#3#4#5{%
  \edef\KVO@param{#5}%
  \@onelevel@sanitize\KVO@param
  \ifx\KVO@param\KVO@true
    \expandafter\@firstofone
  \else
    \ifx\KVO@param\KVO@false
      \expandafter\expandafter\expandafter\@firstofone
    \else
      \ifx#2\@clsextension
        \expandafter\ClassWarning
      \else
        \expandafter\PackageWarning
      \fi
      {#1}{%
        Value `\KVO@param' is not supported by\MessageBreak
        option `#4'%
      }%
      \expandafter\expandafter\expandafter\@gobble
    \fi
  \fi
  {%
    ^^A\ifx#2\@clsextension
    ^^A  \expandafter\ClassInfo
    ^^A\else
    ^^A  \expandafter\PackageInfo
    ^^A\fi
    ^^A{#1}{[option] #4=\KVO@param}%
    \csname#3#4\KVO@param\endcsname
  }%
}
```

`\KVO@true` `\KVO@false` The macros `\KVO@true` and `\KVO@false` are used for string comparisons. After `\@onelevel@sanitize` we have only tokens with catcode 12 (other).

```
\def\KVO@true{true}
\def\KVO@false{false}
\@onelevel@sanitize\KVO@true
\@onelevel@sanitize\KVO@false
```

6.2.3 `\DeclareStringOption`

`\DeclareStringOption`

```
\newcommand*{\DeclareStringOption}[2][]{%
  \@ifnextchar[{%
    \KVO@DeclareStringOption{#1}{#2}@%
  }{%
    \KVO@DeclareStringOption{#1}{#2}{}[]%
  }%
}
```

`\KVO@DeclareStringOption`

```
\def\KVO@DeclareStringOption#1#2#3[#4]{%
  \KVO@ifdefinable{\KVO@prefix#2}{%
    \@namedef{\KVO@prefix#2}{#1}%
    \begingroup
      \ifx\\#3\\%
        \toks@{}%
      \else
        \toks@{[{#4}]}%
      \fi
      \edef\x{\endgroup
        \noexpand\define@key{\KVO@family}{#2}\the\toks@{%
          ^^A\begingroup
          ^^A  \toks@{####1}%
          ^^A  \ifx\@currext\@clsextension
          ^^A    \noexpand\ClassInfo
          ^^A  \else
          ^^A    \noexpand\PackageInfo
          ^^A  \fi
          ^^A  {\@currname}{%
          ^^A    [option] #2={\noexpand\the\toks@}%
          ^^A  }%
          ^^A\endgroup
          \noexpand\def
          \expandafter\noexpand\csname\KVO@prefix#2\endcsname{####1}%
        }%
      }%
    \x
  }%
}
```

6.2.4 \DeclareVoidOption

\DeclareVoidOption

```
\newcommand*{\DeclareVoidOption}[1]{%
  \begingroup
    \let\next\@gobbletwo
    \KVO@ifdefinable{\KVO@prefix#1}{%
      \let\next\@firstofone
    }%
  \expandafter\endgroup
  \next{%
    \begingroup
      \edef\x{\endgroup
        \noexpand\define@key{\KVO@family}{#1}[\KVO@VOID@]{%
          \noexpand\KVO@voidkey{\@currname}%
          \ifx\@currext\@clsextension
            \noexpand\@clsextension
          \else
            \noexpand\@pkgextension
          \fi
          {#1}%
          {####1}%
          \expandafter\noexpand\csname\KVO@prefix#1\endcsname
        }%
      }%
    \x
    \@namedef{\KVO@prefix#1}%
  }%
}
\def\KVO@VOID@{@VOID@}
```

\KVO@voidkey

#1	package/class name
#2	\@pkgextension/\@clsextension
#3	key name
#4	default (@VOID@)
#5	macro with option code

```
\def\KVO@voidkey#1#2#3#4{%
  \def\CurrentOption{#3}%
  \begingroup
    \def\x{#4}%
  \expandafter\endgroup
  \ifx\x\KVO@VOID@
  \else
    \ifx#2\@clsextension
      \expandafter\ClassWarning
    \else
      \expandafter\PackageWarning
    \fi
    {#1}{%
      Unexpected value for option `#3'\MessageBreak
      is ignored%
    }%
  \fi
  ^^A\ifx#2\@clsextension
  ^^A  \expandafter\ClassInfo
  ^^A\else
  ^^A  \expandafter\PackageInfo
  ^^A\fi
  ^^A{#1}{[option] #3}%
}
```

6.2.5 \DeclareDefaultOption

\DeclareDefaultOption

```
\newcommand*{\DeclareDefaultOption}{%
  \@namedef{KVO@default@\@currname.\@currext}%
}
```

6.3 Dynamic options

6.3.1 \DisableKeyvalOption

```
\SetupKeyvalOptions{%
  family=KVOdyn,%
  prefix=KVOdyn@%
}
\DeclareBoolOption[true]{global}
\DeclareComplementaryOption{local}{global}
\DeclareStringOption[undef]{action}
\let\KVOdyn@name\relax
\let\KVOdyn@ext\@empty
\define@key{KVOdyn}{class}{%
  \def\KVOdyn@name{#1}%
  \let\KVOdyn@ext\@clsextension
}
\define@key{KVOdyn}{package}{%
  \def\KVOdyn@name{#1}%
  \let\KVOdyn@ext\@pkgextension
}
\newcommand*{\DisableKeyvalOption}[3][]{%
  \begingroup
    \setkeys{KVOdyn}{#1}%
    \def\x{\endgroup}%
    \@ifundefined{KVO@action@\KVOdyn@action}{%
      \PackageError{kvoptions}{%
        Unknown disable action
        `\expandafter\strip@prefix\meaning\KVOdyn@action'\MessageBreak
        for option `#3' in keyval family '#2'%
      }\@ehc
    }{%
      \csname KVO@action@\KVOdyn@action\endcsname{#2}{#3}%
    }%
  \x
}
\def\KVO@action@undef#1#2{%
  \edef\x{\endgroup
    \ifKVOdyn@global\global\fi
    \let
    \expandafter\noexpand\csname KV@#1@#2\endcsname
    \relax
    \ifKVOdyn@global\global\fi
    \let
    \expandafter\noexpand\csname KV@#1@#2@default\endcsname
    \relax
  }%
  ^^A\PackageInfo{kvoptions}{%
  ^^A  [option] key `#2' of family `#1'\MessageBreak
  ^^A  is disabled (undef, \ifKVOdyn@global global\else local\fi)%
  ^^A}%
}
\def\KVO@action@ignore#1#2{%
  \edef\x{\endgroup
    \ifKVOdyn@global\global\fi
    \let
    \expandafter\noexpand\csname KV@#1@#2\endcsname
    \@gobble
    \ifKVOdyn@global\global\fi
    \let
    \expandafter\noexpand\csname KV@#1@#2@default\endcsname
    \@empty
  }%
  ^^A\PackageInfo{kvoptions}{%
  ^^A  [option] key `#2' of family `#1'\MessageBreak
  ^^A  is disabled (ignore, \ifKVOdyn@global global\else local\fi)%
  ^^A}%
}
\def\KVO@action@error{%
  \KVO@do@action{error}%
}
\def\KVO@action@warning{%
  \KVO@do@action{warning}%
}
```

#1 `error` or `warning`
#2 ⟨*family*⟩
#3 ⟨*key*⟩

```
\def\KVO@do@action#1#2#3{%
  \ifx\KVOdyn@name\relax
    \PackageError{kvoptions}{%
      Action type `#1' needs package/class name\MessageBreak
      for key `#3' in family `#2'%
    }\@ehc
  \else
    \edef\x{\endgroup
      \noexpand\define@key{#2}{#3}[]{%
        \expandafter\noexpand\csname KVO@disable@#1\endcsname
        {\KVOdyn@name}\noexpand\KVOdyn@ext{#3}%
      }%
      \ifKVOdyn@global
        \global\let
        \expandafter\noexpand\csname KV@#2@#3\endcsname
        \expandafter\noexpand\csname KV@#2@#3\endcsname
        \global\let
        \expandafter\noexpand\csname KV@#2@#3@default\endcsname
        \expandafter\noexpand\csname KV@#2@#3@default\endcsname
      \fi
    }%
    ^^A\ifx\KVOdyn@ext\@clsextension
    ^^A  \expandafter\ClassInfo
    ^^A\else
    ^^A   \expandafter\PackageInfo
    ^^A\fi
    ^^A{\KVOdyn@name}{%
    ^^A  [option] key `#3' of family `#2'\MessageBreak
    ^^A  is disabled (#1, \ifKVOdyn@global global\else local\fi)%
    ^^A}%
  \fi
}
\def\KVO@disable@error#1#2#3{%
  \ifx#2\@clsextension
    \expandafter\ClassError
  \else
    \expandafter\PackageError
  \fi
  {#1}{%
    Option `#3' is given too late,\MessageBreak
    now the option is ignored%
  }\@ehc
}
\def\KVO@disable@warning#1#2#3{%
  \ifx#2\@clsextension
    \expandafter\ClassWarning
  \else
    \expandafter\PackageWarning
  \fi
  {#1}{%
    Option `#3' is already consumed\MessageBreak
    and has no effect%
  }%
}
```

6.4 Process options

6.5 \ProcessKeyvalOptions

\ProcessKeyvalOptions If the optional star is given, we get the family name and expand it for safety.

```
\newcommand*{\ProcessKeyvalOptions}{%
  \@ifstar{%
    \begingroup
      \edef\x{\endgroup
        \noexpand\KVO@ProcessKeyvalOptions{\KVO@family}%
      }%
    \x
  }%
  \KVO@ProcessKeyvalOptions
}
```

```
\def\KVO@ProcessKeyvalOptions#1{%
  \let\@tempc\relax
  \let\KVO@temp\@empty
```

Add any global options that are known to KV to the start of the list being built in `\KVO@temp` and mark them used (by removing them from the unused option list).

```
  \ifx\@currext\@clsextension
  \else
    \ifx\@classoptionslist\relax
    \else
      \@for\KVO@CurrentOption:=\@classoptionslist\do{%
        \@ifundefined{KV@#1@\expandafter\KVO@getkey\KVO@CurrentOption=\@nil}{%
        }{%
          \edef\KVO@temp{%
^^B         \unexpanded\expandafter{%
              \KVO@temp
^^B         }%
            ,%
^^B         \unexpanded\expandafter{%
              \KVO@CurrentOption
^^B         }%
            ,%
          }%
^^B       \@onelevel@sanitize\KVO@CurrentOption
          \@expandtwoargs\@removeelement\KVO@CurrentOption
            \@unusedoptionlist\@unusedoptionlist
        }%
      }%
    \fi
  \fi
```

Now stick the package options at the end of the list and wrap in a call to `\setkeys`. A class ignores unknown global options, we must remove them to prevent error messages from `\setkeys`.

```
\begingroup
  \toks\tw@{}%
  \@ifundefined{opt@\@currname.\@currext}{%
    \toks@\expandafter{\KVO@temp}%
  }{%
    \toks@\expandafter\expandafter\expandafter{%
      \csname opt@\@currname.\@currext\endcsname
    }%
    \ifx\@currext\@clsextension
      \edef\CurrentOption{\the\toks@}%
      \toks@\expandafter{\KVO@temp}%
      \@for\CurrentOption:=\CurrentOption\do{%
        \@ifundefined{%
          KV@#1@\expandafter\KVO@getkey\CurrentOption=\@nil
        }{%
```

A class puts not used options in the unused option list.

```
^^B       \@onelevel@sanitize\CurrentOption
          \ifx\@unusedoptionlist\@empty
            \global\let\@unusedoptionlist\CurrentOption
          \else
            \expandafter\expandafter\expandafter\gdef
            \expandafter\expandafter\expandafter\@unusedoptionlist
            \expandafter\expandafter\expandafter{%
              \expandafter\@unusedoptionlist
              \expandafter,\CurrentOption
            }%
          \fi
        }{%
          \toks@\expandafter{%
            \the\expandafter\toks@\expandafter,\CurrentOption
          }%
        }%
      }%
    \else
```

Without default action we pass all options to \setkeys. Otherwise we have to check which options are known. These are passed to \setkeys. For the others the default action is performed.

```
      \@ifundefined{KVO@default@\@currname.\@currext}{%
        \toks@\expandafter\expandafter\expandafter{%
          \expandafter\KVO@temp\the\toks@
        }%
      }{%
        \edef\CurrentOption{\the\toks@}%
        \toks@\expandafter{\KVO@temp}%
        \@for\CurrentOption:=\CurrentOption\do{%
          \@ifundefined{%
            KV@#1@\expandafter\KVO@getkey\CurrentOption=\@nil
          }{%
            \toks\tw@\expandafter{%
              \the\toks\expandafter\tw@\expandafter,\CurrentOption
            }%
          }{%
            \toks@\expandafter{%
              \the\expandafter\toks@\expandafter,\CurrentOption
            }%
          }%
        }%
      }%
    \fi
  }%
  \edef\KVO@temp{\endgroup
```

```
      \noexpand\KVO@calldefault{\the\toks\tw@}%
      \noexpand\setkeys{#1}{\the\toks@}%
    }%
  \KVO@temp
```

Some cleanup of `\ProcessOptions`.

```
  \let\CurrentOption\@empty
  \AtEndOfPackage{\let\@unprocessedoptions\relax}%
}
```

6.5.1 Helper macros

`\KVO@getkey` Extract the key part of a key=value pair.

```
\def\KVO@getkey#1=#2\@nil{#1}
```

`\KVO@calldefault`

```
\def\KVO@calldefault#1{%
  \begingroup
    \def\x{#1}%
  \expandafter\endgroup
  \ifx\x\@empty
  \else
    \@for\CurrentOption:=#1\do{%
      \ifx\CurrentOption\@empty
      \else
        \expandafter\KVO@setcurrents\CurrentOption=\@nil
        \@nameuse{KVO@default@\@currname.\@currext}%
      \fi
    }%
  \fi
}
```

`\KVO@setcurrents` Extract the key part of a key=value pair.

```
\def\KVO@setcurrents#1=#2\@nil{
  \def\CurrentOptionValue{#2}%
  \ifx\CurrentOptionValue\@empty
    \let\CurrentOptionKey\CurrentOption
    \let\CurrentOptionValue\relax
  \else
    \edef\CurrentOptionKey{\zap@space#1 \@empty}%
    \expandafter\KVO@setcurrentvalue\CurrentOption\@nil
  \fi
}
```

`\KV@setcurrentvalue` Here the value part is parsed. Package keyval's `\KV@@sp@def` helps in removing spaces at the begin and end of the value.

```
\def\KVO@setcurrentvalue#1=#2\@nil{%
  \KV@@sp@def\CurrentOptionValue{#2}%
}
```

6.6 Patch

Caution: docstrip stops at `\endinput` at begin of line!

```
^^B\@gobble
          \endinput
\PackageInfo{kvoptions}{Patching LaTeX's option system}
```

Check for ε-TEX.

```
\begingroup\expandafter\expandafter\expandafter\endgroup
\expandafter\ifx\csname eTeXversion\endcsname\relax
  \PackageWarningNoLine{kvoptions}{%
    Option 'patch' ignored, because e-TeX is missing%
  }%
  \expandafter\endinput
\fi
```

Check for package xkvltxp.

```
\@ifpackageloaded{xkvltxp}{%
  \PackageWarningNoLine{kvoptions}{%
    Option ‘patch’ cannot be used together with\MessageBreak
    package ‘xkvltxp’ that is already loaded.\MessageBreak
    Therefore option ‘patch’ will be ignored%
  }%
  \endinput
}{}
```

```
\def\@if@ptions#1#2#3{%
  \begingroup
    \KVO@normalize\KVO@temp{#3}%
    \edef\x{\endgroup
      \noexpand\@if@pti@ns{%
        \detokenize\expandafter\expandafter\expandafter{%
          \csname opt@#2.#1\endcsname
        }%
      }{%
        \detokenize\expandafter{\KVO@temp}%
      }%
    }%
  \x
}
```

```
\def\@pass@ptions#1#2#3{%
  \KVO@normalize\KVO@temp{#2}%
  \@ifundefined{opt@#3.#1}{%
    \expandafter\gdef\csname opt@#3.#1%
          \expandafter\endcsname\expandafter{%
      \KVO@temp
    }%
  }{%
    \expandafter\gdef\csname opt@#3.#1%
          \expandafter\expandafter\expandafter\endcsname
          \expandafter\expandafter\expandafter{%
      \csname opt@#3.#1\expandafter\endcsname\expandafter,\KVO@temp
    }%
  }%
}
```

```
\def\ProcessOptions{%
  \let\ds@\@empty
  \@ifundefined{opt@\@currname.\@currext}{%
    \let\@curroptions\@empty
  }{%
    \expandafter\expandafter\expandafter\def
    \expandafter\expandafter\expandafter\@curroptions
    \expandafter\expandafter\expandafter{%
      \csname opt@\@currname.\@currext\endcsname
    }%
  }%
  \@ifstar\KVO@xprocess@ptions\KVO@process@ptions
}
```

```
\def\KVO@process@ptions{%
  \@for\CurrentOption:=\@declaredoptions\do{%
    \ifx\CurrentOption\@empty
    \else
      \begingroup
        \ifx\@currext\@clsextension
```

```
          \toks@{}%
        \else
          \toks@\expandafter{\@classoptionslist,}%
        \fi
        \toks\tw@\expandafter{\@curroptions}%
        \edef\x{\endgroup
          \noexpand\in@{,\CurrentOption,}{,\the\toks@\the\toks\tw@,}%
        }%
      \x
      \ifin@
        \KVO@use@ption
        \expandafter\let\csname ds@\CurrentOption\endcsname\@empty
      \fi
    \fi
  }%
  \KVO@process@pti@ns
}
\def\KVO@xprocess@ptions{%
  \ifx\@currext\@clsextension
  \else
    \@for\CurrentOption:=\@classoptionslist\do{%
      \ifx\CurrentOption\@empty
      \else
        \KVO@in@\CurrentOption\@declaredoptions
        \ifin@
          \KVO@use@ption
          \expandafter\let\csname ds@\CurrentOption\endcsname\@empty
        \fi
      \fi
    }%
  \fi
  \KVO@process@pti@ns
}
\def\KVO@in@#1#2{%
  \in@false
  \begingroup
    \@for\x:=#2\do{%
      \ifx\x#1\relax
        \in@true
      \fi
    }%
    \edef\x{\endgroup
      \ifin@
        \noexpand\in@true
      \fi
    }%
  \x
}
\def\KVO@process@pti@ns{%
  \@for\CurrentOption:=\@curroptions\do{%
    \@ifundefined{ds@\KVO@SanitizedCurrentOption}{%
      \KVO@use@ption
      \default@ds
    }%
    \KVO@use@ption
  }%
  \@for\CurrentOption:=\@declaredoptions\do{%
    \expandafter\let\csname ds@\CurrentOption\endcsname\relax
  }%
  \let\CurrentOption\@empty
  \let\@fileswith@pti@ns\@@fileswith@pti@ns
  \AtEndOfPackage{\let\@unprocessedoptions\relax}%
```

```
}
\def\KVO@use@ption{%
  \begingroup
    \edef\x{\endgroup
      \noexpand\@removeelement{%
        \detokenize\expandafter{\CurrentOption}%
      }{%
        \detokenize\expandafter{\@unusedoptionlist}%
      }%
    }%
  \x\@unusedoptionlist
  \csname ds@\KVO@SanitizedCurrentOption\endcsname
}
\def\OptionNotUsed{%
  \ifx\@currext\@clsextension
    \xdef\@unusedoptionlist{%
      \ifx\@unusedoptionlist\@empty
      \else
        \detokenize\expandafter{\@unusedoptionlist,}%
      \fi
      \detokenize\expandafter{\CurrentOption}%
    }%
  \fi
}
```

Variant of `\ExecuteOptions` that better protects `\CurrentOption`.

```
\def\CurrentOption@SaveLevel{0}
\def\ExecuteOptions{%
  \expandafter\KVO@ExecuteOptions
      \csname CurrentOption@\CurrentOption@SaveLevel\endcsname
}
\def\KVO@ExecuteOptions#1#2{%
  \let#1\CurrentOption
  \edef\CurrentOption@SaveLevel{\the\numexpr\CurrentOption@SaveLevel+1}%
  \@for\CurrentOption:=#2\do{%
    \csname ds@\CurrentOption\endcsname
  }%
  \edef\CurrentOption@SaveLevel{\the\numexpr\CurrentOption@SaveLevel-1}%
  \let\CurrentOption#1%
}
\def\KVO@fileswith@pti@ns#1[#2]#3[#4]{%
  \ifx#1\@clsextension
    \ifx\@classoptionslist\relax
      \KVO@normalize\KVO@temp{#2}%
      \expandafter\gdef\expandafter\@classoptionslist\expandafter{%
        \KVO@temp
      }%
      \def\reserved@a{%
        \KVO@onefilewithoptions#3[#2][#4]#1%
        \@documentclasshook
      }%
    \else
      \def\reserved@a{%
        \KVO@onefilewithoptions#3[#2][#4]#1%
      }%
    \fi
  \else
    \begingroup
      \let\KVO@temp\relax
      \let\KVO@onefilewithoptions\relax
      \let\@pkgextension\relax
      \def\reserved@b##1,{%
```

```
        \ifx\@nil##1\relax
        \else
          \ifx\relax##1\relax
          \else
            \KVO@onefilewithoptions##1[\KVO@temp][#4]\@pkgextension
          \fi
          \expandafter\reserved@b
        \fi
      }%
      \edef\reserved@a{\zap@space#3 \@empty}%
      \edef\reserved@a{\expandafter\reserved@b\reserved@a,\@nil,}%
      \toks@{#2}%
      \def\KVO@temp{\the\toks@}%
    \edef\reserved@a{\endgroup \reserved@a}%
  \fi
  \reserved@a
}
\def\KVO@onefilewithoptions#1[#2][#3]#4{%
  \@pushfilename
  \xdef\@currname{#1}%
  \global\let\@currext#4%
  \expandafter\let\csname\@currname.\@currext-h@@k\endcsname\@empty
  \let\CurrentOption\@empty
  \@reset@ptions
  \makeatletter
  \def\reserved@a{%
    \@ifl@aded\@currext{#1}{%
      \@if@ptions\@currext{#1}{#2}{%
      }{%
        \begingroup
          \@ifundefined{opt@#1.\@currext}{%
            \def\x{}%
          }{%
            \edef\x{%
              \expandafter\expandafter\expandafter\strip@prefix
              \expandafter\meaning\csname opt@#1.\@currext\endcsname
            }%
          }%
          \def\y{#2}%
          \edef\y{\expandafter\strip@prefix\meaning\y}%
          \@latex@error{Option clash for \@cls@pkg\space #1}{%
            The package #1 has already been loaded
            with options:\MessageBreak
            \space\space[\x]\MessageBreak
            There has now been an attempt to load it
             with options\MessageBreak
            \space\space[\y]\MessageBreak
            Adding the global options:\MessageBreak
            \space\space
                 \x,\y\MessageBreak
            to your \noexpand\documentclass declaration may fix this.%
            \MessageBreak
            Try typing \space <return> \space to proceed.%
          }%
        \endgroup
      }%
    }{%
      \@pass@ptions\@currext{#2}{#1}%
      \global\expandafter
      \let\csname ver@\@currname.\@currext\endcsname\@empty
      \InputIfFileExists
        {\@currname.\@currext}%
        {}%
        {\@missingfileerror\@currname\@currext}%
      \let\@unprocessedoptions\@@unprocessedoptions
      \csname\@currname.\@currext-h@@k\endcsname
      \expandafter\let\csname\@currname.\@currext-h@@k\endcsname
              \@undefined
      \@unprocessedoptions
    }%
    \@ifl@ter\@currext{#1}{#3}{%
    }{%
      \@latex@warning@no@line{%
        You have requested,\on@line,
        version\MessageBreak
          #3' of \@cls@pkg\space #1,\MessageBreak
        but only version\MessageBreak
         `\csname ver@#1.\@currext\endcsname'\MessageBreak
        is available
      }%
    }%
    \ifx\@currext\@clsextension\let\LoadClass\@twoloadclasserror\fi
    \@popfilename
    \@reset@ptions
  }%
  \reserved@a
}
\def\@unknownoptionerror{%
  \@latex@error{%
    Unknown option `\KVO@SanitizedCurrentOption' %
    for \@cls@pkg\space`\@currname'%
  }{%
    The option `\KVO@SanitizedCurrentOption' was not declared in
    \@cls@pkg\space`\@currname', perhaps you\MessageBreak
    misspelled its name.
    Try typing \space <return>
    \space to proceed.%
  }%
}
\def\@@unprocessedoptions{%
  \ifx\@currext\@pkgextension
    \@ifundefined{opt@\@currname.\@currext}{%
       \let\@curroptions\@empty
    }{%
      \expandafter\let\expandafter\@curroptions
          \csname opt@\@currname.\@currext\endcsname
    }%
    \@for\CurrentOption:=\@curroptions\do{%
        \ifx\CurrentOption\@empty\else\@unknownoptionerror\fi
    }%
  \fi
}
\def\KVO@SanitizedCurrentOption{%
  \expandafter\strip@prefix\meaning\CurrentOption
}
```

Normalize option list.

```
\def\KVO@normalize#1#2{%
  \let\KVO@result\@empty
  \KVO@splitcomma#2,\@nil
  \let#1\KVO@result
}
\def\KVO@splitcomma#1,#2\@nil{%
  \KVO@ifempty{#1}{}{%
```

```
    \KVO@checkkv#1=\@nil
  }%
  \KVO@ifempty{#2}{}{\KVO@splitcomma#2\@nil}%
}
\def\KVO@ifempty#1{%
  \expandafter\ifx\expandafter\\\detokenize{#1}\\%
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
}
\def\KVO@checkkv#1=#2\@nil{%
  \KVO@ifempty{#2}{%
    % option without value
    \edef\KVO@x{\zap@space#1 \@empty}%
    \ifx\KVO@x\@empty
      % ignore empty option
    \else
      % append to list
      \edef\KVO@result{%
        \unexpanded\expandafter{\KVO@result},\KVO@x
      }%
    \fi
  }{%
    % #1: "key", #2: "value="
    % add key part
    \edef\KVO@result{%
      \unexpanded\expandafter{\KVO@result},%
      \zap@space#1 \@empty
    }%
    \futurelet\@let@token\KVO@checkfirsttok#2 \@nil| = \@nil|\KVO@nil
  }%
}
\def\KVO@checkfirsttok{%
  \ifx\@let@token\bgroup
    % no space at start
    \expandafter\KVO@removelastspace\expandafter=%
    % "<value><spaceopt>= \@nil"
  \else
    \expandafter\KVO@checkfirstA
  \fi
}
\def\KVO@checkfirstA#1 #2\@nil{%
  \KVO@ifempty{#2}{%
    \KVO@removelastspace=#1 \@nil
  }{%
    \KVO@ifempty{#1}{%
      \KVO@removelastspace=#2\@nil
    }{%
      \KVO@removelastspace=#1 #2\@nil
    }%
  }%
}
\def\KVO@removelastspace#1 = \@nil|#2\KVO@nil{%
  \KVO@ifempty{#2}{%
    \edef\KVO@result{%
      \unexpanded\expandafter{\KVO@result}%
      \unexpanded\expandafter{\KVO@removegarbage#1\KVO@nil}%
    }%
  }{%
    \edef\KVO@result{%
      \unexpanded\expandafter{\KVO@result}%
      \unexpanded{#1}%
    }%
  }%
}
\def\KVO@removegarbage#1= \@nil#2\KVO@nil{#1}%
```

Arguments #1 and #2 are macros.

```
\def\KVO@removeelement#1#2{%
  \begingroup
    \toks@={}%
    \@for\x:=#2\do{%
      \ifx\x\@empty
      \else
        \ifx\x#1\relax
        \else
          \edef\t{\the\toks@}%
          \ifx\t\@empty
          \else
            \toks@\expandafter{\the\toks@,}%
          \fi
          \toks@\expandafter{\the\expandafter\toks@\x}%
        \fi
      \fi
    }%
    \edef\x{\endgroup
      \def\noexpand#2{\the\toks@}%
    }%
  \x
}
```

```
\let\@@fileswith@pti@ns\KVO@fileswith@pti@ns
\ifx\@fileswith@pti@ns\@badrequireerror
\else
  \let\@fileswith@pti@ns\KVO@fileswith@pti@ns
\fi
```

```
⟨/package⟩
```

7 Installation

CTAN. This package is available on CTAN[1]:

CTAN:macros/latex/contrib/oberdiek/kvoptions.dtx The source file.

CTAN:macros/latex/contrib/oberdiek/kvoptions.pdf Documentation.

Unpacking. The .dtx file is a self-extracting docstrip archive. The files are extracted by running the .dtx through plain-TeX:

```
tex kvoptions.dtx
```

TDS. Now the different files must be moved into the different directories in your installation TDS tree (also known as texmf tree):

kvoptions.sty	→	tex/latex/oberdiek/kvoptions.sty
kvoptions.pdf	→	doc/latex/oberdiek/kvoptions.pdf
example-mycolorsetup.sty	→	doc/latex/oberdiek/example-mycolorsetup.sty
kvoptions.dtx	→	source/latex/oberdiek/kvoptions.dtx

If you have a docstrip.cfg that configures and enables docstrip's TDS installing feature, then some files can already be in the right place, see the documentation of docstrip.

[1] ftp://ftp.ctan.org/tex-archive/

Refresh file databases. If your TEX distribution (teTEX, mikTEX, ...) rely on file databases, you must refresh these. For example, teTEX users run `texhash` or `mktexlsr`.

7.1 Some details for the interested

Attached source. The PDF documentation on CTAN also includes the `.dtx` source file. It can be extracted by AcrobatReader 6 or higher. Another option is pdftk, e.g. unpack the file into the current directory:

```
pdftk kvoptions.pdf unpack_files output .
```

Unpacking with LATEX. The `.dtx` chooses its action depending on the format:

plain-TEX: Run docstrip and extract the files.

LATEX: Generate the documentation.

If you insist on using LATEX for docstrip (really, docstrip does not need LATEX), then inform the autodetect routine about your intention:

```
latex \let\install=y\input{kvoptions.dtx}
```

Do not forget to quote the argument according to the demands of your shell.

Generating the documentation. You can use both the `.dtx` or the `.drv` to generate the documentation. The process can be configured by the configuration file `ltxdoc.cfg`. For instance, put this line into this file, if you want to have A4 as paper format:

```
\PassOptionsToClass{a4paper}{article}
```

An example follows how to generate the documentation with pdfLATEX:

```
pdflatex kvoptions.dtx
makeindex -s gind.ist kvoptions.idx
pdflatex kvoptions.dtx
makeindex -s gind.ist kvoptions.idx
pdflatex kvoptions.dtx
```

8 References

[1] Package ifthen, David Carlisle, 2001/05/26. `CTAN:macros/latex/base/ifthen.dtx`

[2] Package helvet, Sebastian Rahtz, Walter Schmidt, 2004/01/26. `CTAN:macros/latex/required/psnfss/psfonts.dtx`

[3] Package hyperref, Sebastian Rahtz, Heiko Oberdiek, 2006/02/12. `CTAN:macros/latex/contrib/hyperref/`

[4] Package keyval, David Carlisle, 1999/03/16. `CTAN:macros/latex/required/graphics/keyval.dtx`

[5] Package multicol, Frank Mittelbach, 2004/02/14. `CTAN:macros/latex/required/tools/multicol.dtx`

[6] Package tabularx, David Carlisle, 1999/01/07. `CTAN:macros/latex/required/tools/tabularx.dtx`

[7] Package tracefnt, Frank Mittelbach, Rainer Schöpf, 1997/05/29. `CTAN:macros/latex/base/ltfsstrc.dtx`

[8] Package xkeyval, Hendri Adriaens, 2005/05/07. CTAN:macros/latex/contrib/xkeyval/

[9] The LaTeX3 Project, *LaTeX 2ε for class and package writers*, 2003/12/09. CTAN:macros/latex/doc/clsguide.pdf

9 History

[0000/00/00 v0.0]

- Probably David Carlisle's code in hyperref was the start.

[2004/02/22 v1.0]

- The first version was never published. It also has offered a patch to get rid of LaTeX's option expansion.

[2006/02/16 v2.0]

- Now the package is redesigned with an easier user interface.
- `\ProcessKeyvalOptions` remains the central service, inherited from hyperref's `\ProcessOptionsWithKV`. Now the use inside classes is also supported.
- Provides help macros for boolean and simple string options.
- Fixes for the patch of LaTeX. The patch is only enabled, if the user requests it.

[2006/02/20 v2.1]

- Unused option list is sanitized to prevent problems with other packages that uses own processing methods for key value options. Disadvantage: the unused global option detection is weakened.
- New option type by `\DeclareVoidOption` for options without value.
- Default rule by `\DeclareDefaultOption`.
- Dynamic options: `\DisableKeyvalOption`.

[2006/06/01 v2.2]

- Fixes for option patch.

[2006/08/17 v2.3]

- `\DeclareBooleanOption` renamed to `\DeclareBoolOption` to avoid a name clash with package `\ifoption`.

[2006/08/22 v2.4]

- Option patch: `\ExecuteOptions` does not change meaning of `\CurrentOption` at all.

10 Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the code line of the definition; numbers in roman refer to the code lines where the entry is used.