xkeyval package

Documentation

Version v1.4, 2004/08/24

Hendri Adriaens
http://stuwww.uvt.nl/~hendri/downloads/xkeyval.html
Center for Economic Research
Tilburg University, the Netherlands

August 25, 2004

Contents

1 Introduction **1**

2 Installation **3**

3 Defining and checking keys **3**

4 Setting keys **4**

5 Pointers **6**

6 Declaring and setting class or package options **9**

7 Error messages **11**

8 Examples **12**

9 Known issues **12**

10 Acknowledgements **13**

11 Copyright **13**

12 Version history **14**

13 Implementation **14**
13.1 TEX program 14
13.2 LATEX program 25
13.3 keyval primitives 28
13.4 TEX header 29

Index **30**

1 Introduction

This package is an extension of the keyval package [2] and offers more flexible macros for defining and setting keys. Both keys defined using the keyval and the xkeyval package can be set by the latter package. The xkeyval macros allow for scanning multiple families for keys. This can, for example, be used to create local families for custom macros and environments which may not access keys meant for other macros and environments, while at the same time, allowing the use of a single command to set all of the keys from the different families globally.

Moreover, the package supplies macros to set up options systems for packages or classes. Options systems created by xkeyval nicely integrate into standard LATEX.

Global options can be copied by packages from the `\documentclass` command and xkeyval deletes `key=value` options from the list of `\documentclass` options to avoid problems for packages which are loaded subsequently and which are not using xkeyval. Packages that do use xkeyval can access the original input to the `\documentclass` command.

The package is compatible to plain TEX and overwrites several keyval macro to provide an easy way to switch between using keyval and xkeyval. This might be useful for package writers that cannot yet rely on the availability of xkeyval in a certain distribution.

After loading xkeyval, loading keyval is prevented to make sure that the extended macros of xkeyval will not be redefined. Some basic keyval macros are supplied in `keyval.tex` to guarantee compatibility to packages that use those macros.

The macros described in section 6 are only available to LATEX users. These macros integrate xkeyval macros into the LATEX options system. To load xkeyval, TEX users do `\input xkeyval.tex`. LATEX users do `\usepackage{xkeyval}` or `\RequirePackage{xkeyval}`. It is mandatory for LATEX users to load xkeyval at any point after the `\documentclass` command. Loading xkeyval from the class which is the document class itself is possible.

In this documentation, often will be referred to the term 'options list'. This is a list of the form:

Example

```
keya= test a, keyb={test b,c,d}, ,keyc , keyd=end
```

From values consisting entirely of a `{ }` group, the outer braces will be stripped off. This allows the user to 'hide' any commas or equality signs that appear in the value of a key. This means that when using braces, xkeyval will not terminate the `key=value` pair when it encounters a comma. For instance, see the value of `keyb` in the example above. Notice further that any white space around '`=`' and '`,`' is ignored. Finally, `keyc` did not get a value. If no default value has been defined for this key, an error will be generated. More details about this can be found in sections 3, 4 and 5.

Throughout this documentation, you will find some examples with a short description. More examples can be found in some example files that come with this package. See section 8 for more information.

Note that the macros `\setkeys`, `\setrmkeys` and `\ExecuteoptionsX` can take fragile commands as values for keys. These will only be expanded at the moment that keys are actually set. Setting keys can be postponed by using `\setkeys*` and `\setrmkeys` or by defining keys that define a macro which can be used later. The macro `\ProcessOptionsX` cannot protect fragile commands. The reason is,

that this macro gets the options specified by the user from LaTeX which already expands fragile commands.

Note finally, that both `[`⟨*arg*⟩`]` and `<`⟨*arg*⟩`>` denote optional arguments to macros for this package. The package uses this syntax to identify the different optional arguments when they appear next to each other.

2 Installation

This package is included in the MiKTeX distribution and can be installed for you by this program. It is also available from CTAN (`http://www.ctan.org/tex-archive/macros/generic/xkeyval`) and can also be installed manually. The package includes pre-generated run and doc files, but you can reproduce them from the source if necessary. See the first lines of `xkeyval.dtx` for information on how to do this. The files `keyval.tex`, `xkeyval.def`, `xkeyval.tex` and `xkeyval.sty` should go into a directory searched by TeX or LaTeX. See the documentation of your LaTeX distribution or the TeX Frequently Asked Questions [4] for more information on installing xkeyval into your LaTeX distribution.

3 Defining and checking keys

This section describes how to define keys and check whether keys already exist.

`\define@key[`⟨*prefix*⟩`]{`⟨*family*⟩`}{`⟨*key*⟩`}[`⟨*default*⟩`]{`⟨*function*⟩`}`

This defines a macro `\prefix@family@key` with one argument holding ⟨*function*⟩. The default value for ⟨*prefix*⟩ is `KV`. This is the standard throughout the package to simplify mixing keyval and xkeyval keys. When ⟨*key*⟩ is used in an options list containing `key=value`, the macro `\prefix@family@key` receives `value` as its argument. The argument can be accessed by ⟨*function*⟩ by using `#1` inside the function.

Example

```
\define@key{family}{key}{The input is: #1}
```

xkeyval will generate an error when the user omits `=value` for a key in the options list. To avoid this, the optional argument can be used to specify a default value.

Example

```
\define@key{family}{key}[none]{The input is: #1}
```

This will additionally define a macro `\prefix@family@key@default` as a macro with no arguments and definition `\prefix@family@key{none}` which will be used when `=value` is missing for `key`.

Keys can also be used to define macros which can be used later.

Example

```
\define@key{family}{key}{\def\mymacro{#1}}
```

When ⟨*prefix*⟩ is specified and empty, the macros created by `\define@key` will have the form `\family@key`. When ⟨*family*⟩ is empty, the resulting form will be `\prefix@key`. When both ⟨*prefix*⟩ and ⟨*family*⟩ are empty, the form is `\key`.

The intended use for ⟨*family*⟩ is to create distinct sets of keys. This can be used to avoid a macro setting keys meant for another macro only. The optional ⟨*prefix*⟩ can be used to identify keys specifically for your package. Using a package specific prefix reduces the probability of multiple packages defining the same key macros. This optional argument can also be used to set keys of some existing packages which use a system based on keyval[1].

To further reduce the probability of redefining existing macros using `\define@key`, xkeyval can perform some checks.

`\define@key*[`⟨*prefix*⟩`]{`⟨*family*⟩`}{`⟨*key*⟩`}[`⟨*default*⟩`]{`⟨*function*⟩`}`

This macro works just as `\define@key` except that it first checks whether `\prefix@family@key` has already been defined. If this is the case, it will produce an error. You might use this when debugging your package, but it is advised to use `\define@key` when releasing your package to avoid incomprehensible errors for users of your package to appear, for instance when an earlier loaded package uses the same key macros as your package (which you might not have known). These errors can generally be avoided however by selecting a proper prefix for your package macros and keys.

`\XKV@ifku[`⟨*prefix*⟩`]{`⟨*families*⟩`}{`⟨*key*⟩`}{`⟨*undefined*⟩`}{`⟨*defined*⟩`}`

This macro executes ⟨*undefined*⟩ when ⟨*key*⟩ is not defined in a family listed in ⟨*families*⟩ using ⟨*prefix*⟩ (which is `KV` by default) and ⟨*defined*⟩ when it is.

Example

```
\XKV@ifku{familya,familyb}{keya}{‘keya’ not defined}{‘keya’ defined}
```

4 Setting keys

This section describes the available macros for setting keys. All of the macros in this section have an optional argument ⟨*prefix*⟩ which determines part of the form of the keys that the macros will be looking for. See section 3. This optional argument takes the value `KV` be default. The examples in this section assume the existence of some keys, defined as, for instance

[1] Like PSTricks, which uses a system originating from keyval, but which has been modified to use no families and `psset` as prefix.

Example

```
\define@key[my]{familya}{keya}{#1}
\define@key[my]{familya}{keyb}{#1}
\define@key[my]{familyb}{keyb}{#1}
```

`\setkeys[`⟨*prefix*⟩`]{`⟨*families*⟩`}[`⟨*na*⟩`]{`⟨*keys*⟩`}`

This macro sets keys of the form `\prefix@family@key` where `family` is an element of the list ⟨*families*⟩ and `key` is an element of the options list ⟨*keys*⟩ and not of ⟨*na*⟩. The latter list can be used to specify keys that should be ignored by the macro. If a key is defined by more families in the list ⟨*families*⟩, the first family from the list defining the key will set it. No errors are produced when ⟨*keys*⟩ is empty. If `family` is empty, the macro will set keys of the form `\prefix@key`. If ⟨*prefix*⟩ is specified and empty, the macro will set keys of the form `\family@key`. If both ⟨*prefix*⟩ and `family` are empty, the macro will set keys of the form `\key`. This is in line with how key macros are constructed (see section 3).

Example

```
\setkeys[my]{familya,familyb}{keya=test}
\setkeys[my]{familya,familyb}{keyb=test}
\setkeys[my]{familyb,familya}{keyb=test}
```

In the example above, line 1 will set `keya` in family `familya`. The next line will set `keyb` in `familya`. The last one sets `keyb` in family `familyb`.

`\setkeys*[`⟨*prefix*⟩`]{`⟨*families*⟩`}[`⟨*na*⟩`]{`⟨*keys*⟩`}`

The stared version of `\setkeys` sets keys which it can locate in the families given and will not produce errors when it cannot find a key. Instead, these keys and their values will be appended to a list of remaining keys in the macro `\XKV@rm` after the use of `\setkeys*`. Keys listed in ⟨*na*⟩ will be ignored fully and will not be appended to the `\XKV@rm` list.

Example

```
\setkeys*[my]{familyb}{keya=test}
```

Since `keya` is not defined in `familyb`, the value in the example above will be stored for later use and no errors are raised.

`\setrmkeys[`⟨*prefix*⟩`]{`⟨*families*⟩`}[`⟨*na*⟩`]`

The macro `\setrmkeys` sets the remaining keys given by the list `\XKV@rm` stored previously by a `\setkeys*` (or `\setrmkeys*`) command in ⟨*families*⟩. ⟨*na*⟩ again lists keys that should be ignored. It will produce an error when a key cannot be located.

Example

```
\setrmkeys[my]{familya}
```

This submits `keya=test` from the previous `\setkeys*` command to `familya`. `keya` will be set.

`\setrmkeys*[`⟨*prefix*⟩`]{`⟨*families*⟩`}[`⟨*na*⟩`]`

The macro `\setrmkeys*` acts as the `\setrmkeys` macro but now, as with `\setkeys*`, it ignores keys that it cannot find and puts them again on the list stored in `\XKV@rm`. Keys listed in ⟨*na*⟩ will be ignored fully and will not be appended to the list in `\XKV@rm`.

Example

```
\setkeys*[my]{familyb}{keya=test}
\setrmkeys*[my]{familyb}
\setrmkeys[my]{familya}
```

In the example above, the second line tries to set `keya` in `familyb` again and no errors are generated on failure. The last line finally sets `keya`.

`\setkeys+[`⟨*prefix*⟩`]{`⟨*families*⟩`}[`⟨*na*⟩`]{`⟨*keys*⟩`}`
`\setkeys*+[`⟨*prefix*⟩`]{`⟨*families*⟩`}[`⟨*na*⟩`]{`⟨*keys*⟩`}`
`\setrmkeys+[`⟨*prefix*⟩`]{`⟨*families*⟩`}[`⟨*na*⟩`]`
`\setrmkeys*+[`⟨*prefix*⟩`]{`⟨*families*⟩`}[`⟨*na*⟩`]`

These macros act as their counterparts without the `+`. However, when a key in ⟨*keys*⟩ is defined by multiple families, this key will be set in *all* families in ⟨*families*⟩. This can, for instance, be used to set keys defined by your own package and by another package with the same name but in different families with a single command.

Example

```
\setkeys+[my]{familya,familyb}{keyb=test}
```

The example above sets `keyb` in both families. See also section 8 for more examples.

5 Pointers

The xkeyval package allows to use pointers in key values. These pointers can point to a key for which previously a value has been recorded. The value for a key named `key` will be stored in the macro `\XKV@key@value`. The syntax of a pointer is `^{pointer}`. The following example will demonstrate how to use pointers (using the keys defined in section 4).

Example

```
\recordkeyvalstrue
\setkeys[my]{familya}{keya=test}
\recordkeyvalsfalse
\setkeys[my]{familyb}{keyb=^{keya}}
```

The value submitted to `keyb` points to `keya`. This has the effect that the value recorded for `keya` will replace `^{keya}` and this value (here `test`) will be submitted to the key macro `keyb`. Notice that recording key values in macros can consume big amounts of memory and hence, recording is turned off by default. You can turn it on by `\recordkeyvalstrue` and off again `\recordkeyvalsfalse` when you don't need to record key values anymore. An error will be raised in the case that a key value points to a key for which no value has been stored.

It is possible to nest pointers as the next example shows.

Example

```
\recordkeyvalstrue
\setkeys[my]{familya}{keya=test}
\setkeys[my]{familyb}{keyb=^{keya}}
\recordkeyvalsfalse
\setkeys[my]{familya}{keyb=^{keyb}}
```

This works as follows. First xkeyval records the value `test` in a macro based on the key name, `keya`, but not on the family name. Then, `keyb` uses that value. Besides that, the value submitted to `keyb`, namely `^{keya}` will be recorded in another macro. Finally, `keyb` from family `familya` will use the value recorded previously for `keyb`, namely `^{keya}`. That in turn points to the value saved for `keya` and that value will be used.

A word of caution is necessary here. You might get into an infinite loop if pointers are not applied with care, as the example below shows.

Example

```
\recordkeyvalstrue
\setkeys[my]{familya}{keya=^{keya}}
```

Luckily, TEX will immediately warn you with the message `TeX capacity exceeded` when this is happening. At this point, you might be wondering why the previous example actually worked. If we have a quick look at that example again, we see that `keyb=^{keyb}` works there since a value has been saved for a key named `keyb` and because recording of key values has been turned off before issuing the command. This avoids xkeyval overwriting the saved value for `keyb`, namely `^{keya}`, by `^{keyb}`. This is another reason to turn off key value recording as soon as you don't need it anymore.

It should also be noted that pointers cannot be read from inside grouped material, `{...}`, if this group is not around the entire key (since that will be stripped off, see section 1). The following, for instance, will not work.

Example

```
\recordkeyvalstrue
\setkeys[my]{familya}{keya=test}
\recordkeyvalfalse
\setkeys[my]{familyb}{keyb=\parbox{2cm}{^{keya}}}
```

The following is a working alternative for this situation.

Example

```
\recordkeyvalstrue
\setkeys[my]{familya}{keya=test}
\recordkeyvalfalse
\setkeys[my]{familyb}{keyb=\begin{minipage}{2cm}^{keya}\end{minipage}}
```

In case there is no appropriate alternative, we can work around the restriction, for instance by using a value macro directly

Example

```
\recordkeyvalstrue
\setkeys[my]{familya}{keya=test}
\recordkeyvalfalse
\setkeys[my]{familyb}{keyb=\parbox{2cm}{\XKV@keya@value}}
```

or by doing

Example

```
\define@key[my]{tempfamily}{tempkey}{\def\tempval{#1}}
\recordkeyvalstrue
\setkeys[my]{familya}{keya=test}
\recordkeyvalfalse
\setkeys[my]{tempfamily}{tempkey=^{keya}}
\setkeys[my]{familyb}{keyb=\parbox{2cm}{\tempval}}
```

Pointers can also be used in default values. We finish this section with an example which demonstrates this.

Example

```
\define@key{fam}{keya}{keya: #1\par}
\define@key{fam}{keyb}[^{keya}]{keyb: #1\par}
\define@key{fam}{keyc}[^{keyb}]{keyc: #1\par}
\recordkeyvalstrue
\setkeys{fam}{keya=test}
\setkeys{fam}{keyb}
\recordkeyvalsfalse
\setkeys{fam}{keyc}
```

6 Declaring and setting class or package options

The macros in this section can be used to build LaTeX class or package options systems using xkeyval. They are comparable to the standard LaTeX macros without the trailing `X`. See for more information about these LaTeX macros the documentation of the source [1] or a LaTeX manual (for instance, the LaTeX Companion [3]). The macros in this section have been built using `\define@key` and `\setkeys` and are not available to TeX users.

The macros below allow for specifying the ⟨*family*⟩ (or ⟨*families*⟩), on which the macros should act, as an optional argument. This could be useful if you want to define global options which can reused later (and set locally by the user) in a macro or environment that you define. If no ⟨*family*⟩ (or ⟨*families*⟩) is specified, the macro will insert the default family name which is the filename of the file that is calling the macros. The macros in this section also allow for setting an optional prefix. When using the filename as family, uniqueness of key macros is already guaranteed. In that case, you can omit the optional ⟨*prefix*⟩. However, when you use a custom prefix for other keys in your package and you want to be able to reset all of the keys later with a single command, you can use the custom prefix also for the class or package options system.

At loading, xkeyval performs two actions. These require xkeyval to be loaded at any point after `\documentclass` or by the document class itself. First, it retrieves the document class and stores that (including the class extension) into the following macro.

`\XKV@documentclass`

This macro could for instance contain `article.cls` and can be useful when using `\ProcessOptionsX*` in a class. See page 11.

Second, the global options stored in `\@classoptionslist` are copied to the following macro.

`\XKV@classoptionslist`

This macro will be used by `\ProcessOptionsX`. Options containing an equality sign are deleted from the original list in `\@classoptionslist` to avoid packages, which do not use xkeyval and which are loaded later, running into problems when trying to copy global options using LaTeX's `\ProcessOptions`.

`\DeclareOptionX[`⟨*prefix*⟩`]<`⟨*family*⟩`>{`⟨*key*⟩`}[`⟨*default*⟩`]{`⟨*function*⟩`}`

Declares an option (i.e., a key). This macro is comparable to the standard LaTeX macro `\DeclareOption`, but with this command, the user can pass a value to the option as well. Reading that value can be done by using `#1` in ⟨*function*⟩. This will contain ⟨*default*⟩ when no value has been specified for the key. The value of the optional argument ⟨*default*⟩ is empty by default. This implies that when the

user does not assign a value to ⟨*key*⟩ and when no default value has been defined, no error will be produced. The optional argument ⟨*family*⟩ can be used to specify a custom family for the key. When the argument is not used, the macro will insert the default family name.

Example

```
\newif\iflandscape
\DeclareOptionX{landscape}{\landscapetrue}
\DeclareOptionX{parindent}{\setlength{\parindent}{#1}}
```

The first option does not use the value that might have been submitted to it. The second option uses the value to set `\parindent`.

`\DeclareOptionX*{`⟨*function*⟩`}`

This macro can be used to process any unknown inputs. It is comparable to the LaTeX macro `\DeclareOption*`. Use `\CurrentOption` within this macro to get the entire input from which the key is unknown, for instance `unknownkey=value` or `somevalue`. These values (possibly including a key) could for example be passed on to another class or package or could be used as an extra class or package option specifying for instance a style that should be loaded.

Example

```
\DeclareOptionX*{\PackageWarning{mypackage}{`\Currentoption' ignored}}
```

The example produces a warning when the user issues an option that has not been declared.

`\ExecuteOptionsX[`⟨*prefix*⟩`]<`⟨*families*⟩`>[`⟨*na*⟩`]{`⟨*keys*⟩`}`

This macro sets keys created by `\DeclareOptionX`. The optional argument ⟨*na*⟩ specifies keys that should be ignored. The optional argument ⟨*families*⟩ can be used to specify a list of families which define ⟨*keys*⟩. When the argument is not used, the macro will insert the default family name. Notice that when `\ExecuteOptionsX` is called in between `\DeclareOptionX` and `\ProcessOptionsX` commands and if `\DeclareOptionX*` has been used, unknown keys in `\ExecuteOptionsX` will be set using the macro created by `\DeclareOptionX*`.

Example

```
\ExecuteOptionsX{parindent=0pt}
```

This initializes `\parindent`.

`\ProcessOptionsX[`⟨*prefix*⟩`]<`⟨*families*⟩`>[`⟨*na*⟩`]`

This macro processes the keys and values passed by the user to the class or package. The optional argument ⟨*na*⟩ can be used to specify keys that should be ignored.

The optional argument ⟨*families*⟩ can be used to specify the families that have been used to define the keys. Note that this macro will not protect fragile user inputs (like `\thepage`) as explained in section 1. When used in a class file, this macro will ignore unknown keys or options (using `\setkeys*`). This allows the user to use global options in the `\documentclass` command which can be copied by packages loaded afterwards.

`\ProcessOptionsX*[`⟨*prefix*⟩`]<`⟨*families*⟩`>[`⟨*na*⟩`]`

The stared version works like `\ProcessOptionsX` except that it also copies user input from the `\documentclass` command. When the user specifies an option in the document class which also exists in the local family (or families) of the package issuing `\ProcessOptionsX*`, the local key will be set as well. In this case, `#1` in the `\DeclareOptionX` macro will contain the value entered in the `\documentclass` command for this key. First the global options from `\documentclass` will set local keys and afterwards, the local options, specified with `\usepackage`, `\RequirePackage` or `\LoadClass` (or friends), will set local keys, which could overwrite the global options again, depending on the way the options sections are constructed. This macro reduces to `\ProcessOptionsX` only when issued from the class which forms the document class for the file at hand to avoid setting the same options twice, but not for classes loaded later using for instance `\LoadClass`. Class options that do not have a counterpart in local families will be skipped.

The use of `\ProcessOptionsX*` in a class file might be tricky since the class could also be used as a basis for another package or class using `\LoadClass`. In that case, depending on the options system of the document class, the behavior of the class loaded with `\LoadClass` could change compared to the situation when it is loaded by `\documentclass`. But since it is technically possible to create two classes that cooperate, the xkeyval package allows for the usage of `\ProcessOptionsX*` in class files. Notice that using LaTeX's `\ProcessOptions` or `\ProcessOptions*`, a class file cannot copy document class options.

In case you want to verify whether your class is loaded with `\documentclass` or `\LoadClass`, you can use the `\XKV@documentclass` macro which contains the current document class.

7 Error messages

There are several points where xkeyval performs a check and could produce an error. These points are listed below. The macros producing the error are listed in between brackets. Numbers of code lines have been supplied for the interested reader. Refer to section 13 for the source code documentation.

1) A key has already been defined (`\setkeys*`). See code line 132.

2) An option or key is not defined (`\setkeys`, `\setrmkeys`, `\ExecuteOptionsX`, `\ProcessOptionsX` or `\ProcessOptionsX*`). See code lines 166, 172 and 436.

3) A value has been given, but no key (idem). See code line 193.

4) No value has been recorded for a key to which a pointer was used (idem). See code line 214.

5) No value is given for the key and no default value has been defined (idem). See code lines 251 and 450.

6) xkeyval is loaded before `\documentclass`. This concerns only LaTeX users. See code line 355.

7) `\RequirePackage` or `\LoadClass` is used within the options section of a package or class. This concerns only LaTeX users. See code lines 368 and 381.

8 Examples

This package includes a zip-file `xkvex.zip` which contains a number of example files. The file `xkvex1.tex` provides an example for LaTeX users for the macros described in sections 3, 4 and 5. The file `xkvexTeX.tex` provides an example for TeX users for the same macros. The files `xkvex2.tex`, `xkveca.cls`, `xkvecb.cls`, `xkvesa.sty`, `xkvesb.sty` and `xkvesc.sty` together form an example for the macros described in section 6. These also demonstrate the possibilities of interaction between packages or classes not using xkeyval and packages or classes that do use xkeyval to set options.

9 Known issues

This package redefines keyval's `\define@key` and `\setkeys`. This is risky in general. However, since xkeyval extends the possibilities of these commands while still allowing for the keyval syntax and use, there should be no problems for packages using these commands after loading xkeyval. The package prevents keyval to be loaded afterwards to avoid these commands from being redefined again into the simpler versions. For packages using internals of keyval, like `\KV@errx` and `\KV@do`, these are provided separately in `keyval.tex`.

The advantage of redefining these commands instead of making new commands is that it is much easier for package authors to start using xkeyval instead of keyval. Further, it eliminates the confusion of having multiple commands doing similar things.

A potential problem lies in other packages that redefine either `\define@key` or `\setkeys` or both. Hence particular care has been spend to check packages for this. Only one package has been found to do this, namely pst-key. This package implements a custom version of `\setkeys` which is specialized to set PSTricks

keys of the form `\psset@somekey`. xkeyval also provides the means to set these kind of keys (see page 4) and work is going on to convert PSTricks packages to be using a specialization of xkeyval instead of pst-key. However, since a lot of authors are involved and since it requires a change of policy, this might take some time. Hence, at the moment of writing, xkeyval will conflict with pst-key and the PSTricks packages using pst-key, which are pst-3dplot, pst-abspos, pst-circ, pst-eucl, pst-fr3d, pst-geo, pst-gr3d, pst-labo, pst-lens, pst-ob3d, pst-optic, pst-osci, pst-poly, pst-stru, pst-uml and pst-vue3d.

10 Acknowledgements

I want to thank Josselin Noirel, Till Tantau and Herbert Voß for help and suggestions. I want to thank Donald Arseneau for contributing the `\@ifnextcharacter` macro. Special thanks go to Uwe Kern for his ideas for improving the functionality of this package, a lot of useful comments on the package and the documentation and for contributing the `\@selective@sanitize` macro.

References

[1] Johannes Braams, David Carlisle, Alan Jeffrey, Leslie Lamport, Frank Mittelbach, Chris Rowley, and Rainer Schöpf. *The LaTeX 2ε Sources*. `http://www.ctan.org/tex-archive/macros/latex/base`, 2003.

[2] David Carlisle. *keyval*. `http://www.ctan.org/tex-archive/macros/latex/required/graphics`, 1999.

[3] Frank Mittelbach and Michel Goosens, with Johannes Braams, David Carlisle, and Chris Rowley. *The LaTeX Companion, Second Edition*. Addison-Wesley, 2004.

[4] TeX FAQ – Installing packages. `http://www.tex.ac.uk/cgi-bin/texfaq2html?label=instpackages`.

11 Copyright

12 Version history

v1.0 *(2004/04/29)*
General: Initial release 1
v1.1 *(2004/30/04)*
`\XKV@d@x`: Made to insert an empty default value if none present for `\DeclareOptionX` 26
v1.2 *(2004/05/08)*
General: Change to `\DeclareOptionXf`; macro is now replaced 1
v1.3 *(2004/05/09)*
General: Moved the options section to the end of the package to allow it to use xkeyval option macros 28
Revision of documentation . . . 1
v1.4 *(2004/08/24)*
`\@ifnextcharacter`: Added robust next character check . . 15
General: Added keyval primitives 28
Added + option to macros . . . 6
Added pointer syntax 6
Added prefix options to macros 3, 4, 9
Changed package options 11
Made package TEX compatible 1
Renamed macros to keyval names 1
`\define@key`: Added optional check 18
`\ProcessOptionsX`: Fixed macro for `\LoadClass` case 27
`\XKV@filterclassoptions`: Fixed small bug 25
`\XKV@split`: Made macro more efficient 20

13 Implementation

13.1 TEX program

Adjust some catcodes to safely define macros.

```
%<*tex>
\edef\XKVAtCode{\the\catcode`\@}
\edef\XKVHatCode{\the\catcode`\^}
\catcode`\@=11\relax
\catcode`\^=12\relax
```

Load LATEX primitives if necessary.

```
\ifx\XKeyValLoaded\@undefined
  \message{xkeyval: key=value parser, v1.4, 2004/08/24 (HA)}
  \input xkeyval.def
\fi
```

Check whether keyval has been loaded and if not, load keyval primitives, define `\KV@err` and `\KV@errx` which are used by several packages and prevent keyval from being loaded after xkeyval.

```
\expandafter\ifx\csname ver@keyval.sty\endcsname\relax
  \input keyval.tex
  \def\KV@errx#1{\XKV@err{#1}}
  \let\KV@err\KV@errx
  \expandafter\def\csname ver@keyval.sty\endcsname{1999/03/16}
\fi
```

Initializations.

```
\newif\ifXKV@st
\newif\ifXKV@pl
\newif\ifXKV@knf
\let\XKV@rm\empty
```

`\ifreckeyvals` This if can be used to force `\setkeys` and `\setrmkeys` to save values submitted to keys.

```
\newif\ifrecordkeyvals
```

`\@ifnextcharacter` `\@ifncharacter` Check the next character independently of its catcode. This will be used to safely perform `\@ifnextcharacter+` and `\@ifnextcharacter*`. This avoids errors in case any other package changes the catcode of these characters.
Contributed by Donald Arseneau.

```
\long\def\@ifnextcharacter#1#2#3{%
  \@ifnextchar\bgroup
  {\@ifnextchar{#1}{#2}{#3}}%
  {\@ifncharacter{#1}{#2}{#3}}%
}
\long\def\@ifncharacter#1#2#3#4{%
  \if\string#1\string#4%
    \expandafter\@firstoftwo
  \else
    \expandafter\@secondoftwo
  \fi
  {#2}{#3}#4%
}
```

`\@selective@sanitize` `{`⟨*character string*⟩`}{`⟨*cmd*⟩`}`
Converts selected characters, given by ⟨*character string*⟩, within the first-level expansion of ⟨*cmd*⟩ to category code 12, leaving all other tokens (including grouping braces) untouched. Thus, macros inside ⟨*cmd*⟩ do not lose their function, as it is the case with `\@onelevel@sanitize`. The resulting token list is again saved in ⟨*cmd*⟩.
Example: `\def\cs{ ^{\fi}~}` and `\@selective@sanitize{!^}\cs` will change the catcode of '`^`' to *other* within `\cs`, while `\fi` and '`~`' will remain unchanged. As the example shows, unbalanced conditionals are allowed.
Remarks: ⟨*cmd*⟩ should not contain the control sequence `\bgroup`; however, `\csname bgroup\endcsname` and `\egroup` are possible.
Contributed by Uwe Kern.

```
\def\@selective@sanitize#1#2%
 {\begingroup
   \toks@\expandafter{#2}%
   \def#2{#1}\@onelevel@sanitize#2%
   \edef#2{{#2}{\the\toks@}}%
   \expandafter\@@selective@sanitize\expandafter#2#2%
  \expandafter\endgroup\expandafter\def\expandafter#2\expandafter{#2}}
```

`\@@selective@sanitize` `{`⟨*cmd*⟩`}{`⟨*sanitized character string*⟩`}{`⟨*token list*⟩`}`

Performs the main work. Here, the characters in ⟨*sanitized character string*⟩ are already converted to catcode 12, ⟨*token list*⟩ is the first-level expansion of the original contents of ⟨*cmd*⟩. The macro basically steps through the ⟨*token list*⟩, inspecting each single token to decide whether it has to be sanitized or passed to the result list. Special care has to be taken to detect spaces, grouping characters and conditionals (the latter may disturb other expressions). However, it is easier and more efficient to look for TEX primitives in general — which are characterized by a `\meaning` that starts with a backslash — than to test whether a token equals specifically `\if`, `\else`, `\fi`, etc. Note that `\@@selective@sanitize` is being called recursively if ⟨*token list*⟩ contains grouping braces.

```
\def\@@selective@sanitize#1#2#3%
 {\def\@i{\futurelet\@@tok\@ii}%
  \def\@ii
   {\expandafter\@iii\meaning\@@tok\relax
    \ifx\@@tok\@@selective@sanitize
      \let\@@cmd\@gobble
    \else
      \ifx\@@tok\@sptoken
        \toks@\expandafter{#1}\edef#1{\the\toks@\space}%
        \def\@@cmd{\afterassignment\@i\let\@@tok= }%
      \else
        \let\@@cmd\@iv
      \fi
    \fi\@@cmd}%
  \def\@iii##1##2\relax{\if##1\@backslashchar\let\@@tok\relax\fi}%
  \def\@iv##1%
   {\toks@\expandafter{#1}\@temptokena{##1}%
    \ifx\@@tok\bgroup
      \begingroup
       \def#1{\expandafter\@@selective@sanitize
        \csname\string#1\endcsname{#2}}%
       \expandafter#1\expandafter{\the\@temptokena}%
       \expandafter\@temptokena\expandafter\expandafter\expandafter
        {\csname\string#1\endcsname}%
      \expandafter\endgroup\edef#1{\the\toks@{\the\@temptokena}}%
      \let\@@cmd\@i
    \else
      \edef#1{\expandafter\string\the\@temptokena}%
      \expandafter\in@\expandafter{#1}{#2}%
      \edef#1{\the\toks@\ifin@#1\else
              \ifx\@@tok\@sptoken\space\else\the\@temptokena\fi\fi}%
      \edef\@@cmd{\noexpand\@i\ifx\@@tok\@sptoken\the\@temptokena\fi}%
    \fi
    \@@cmd}%
  \let#1\@empty \@i#3\@@selective@sanitize}%
```

`\XKV@err` Error macro.

```
\def\XKV@err#1{\errmessage{xkeyval error: #1}}
```

`\XKV@ifstar` `\XKV@ifplus` Checks whether the following token is a `*` or `+`. Use `\XKV@ifnextchar` to perform the action safely.

```
\def\XKV@ifstar#1{\@ifnextcharacter*{\@firstoftwo{#1}}}
\def\XKV@ifplus#1{\@ifnextcharacter+{\@firstoftwo{#1}}}
```

`\XKV@@zs@def` ⟨*cmd*⟩⟨*token*⟩
Defines ⟨*cmd*⟩ as ⟨*token*⟩ after expanding ⟨*token*⟩ and deleting any spaces within ⟨*token*⟩.

```
\def\XKV@@zs@def#1#2{\edef#1{\expandafter\zap@space#2 \empty}}
```

`\XKV@whilist` ⟨*cmd*⟩`:=`⟨*list*⟩`\do`⟨*if*⟩`\fi`⟨*function*⟩
Based on `\@for`. The macro executes ⟨*function*⟩ while ⟨*if*⟩ is valid. At every iteration, the first element will be taken from ⟨*list*⟩ and ⟨*cmd*⟩ will be defined to expand to this element. Execution stops when the list has ran out of elements. Hence this macro is a combination of `\@for` and `\@whilesw`. When using `\iftrue` for ⟨*if*⟩, the execution of the macro is the same as that of `\@for`, but contains an additional check and is hence less efficient than `\@for` in that situation. This macro does not use temporary macros.

```
\long\def\XKV@whilist#1:=#2\do#3\fi#4{%
```

Check whether the list is non-empty and the condition true and start iteration.

```
  \ifx\empty#2\empty\else
    #3\expandafter\XKV@wh@list#2,\@nil,\@nil\@@#1#3\fi{#4}\fi
  \fi
}
```

`\XKV@wh@list` Performs iteration and checks extra condition. This macro is not optimized for the case that the list contains a single element.

```
\long\def\XKV@wh@list#1,#2\@@#3#4\fi#5{%
```

Define the running ⟨*cmd*⟩.

```
  \def#3{#1}%
```

If we find the end of the list, stop.

```
  \ifx#3\@nnil
    \expandafter\XKV@wh@l@st
  \else
```

If the condition is met, execute ⟨*function*⟩ and continue. Otherwise stop.

```
    #4
      #5\expandafter\expandafter\expandafter\XKV@wh@list
    \else
      \expandafter\expandafter\expandafter\XKV@wh@l@st
    \fi
  \fi
  #2\@@#3#4\fi{#5}%
}
```

`\XKV@wh@l@st` Macro to gobble remaining input.

```
\long\def\XKV@wh@l@st#1\@@#2#3\fi#4{}
```

`\XKV@makepf` This macro creates the prefix, like `prefix@` in `\prefix@family@key`. First it deletes spaces from the input and checks wether it is empty. If not empty, an `@`-sign is added.

```
\def\XKV@makepf#1{%
  \XKV@@zs@def\XKV@prefix{#1}%
  \ifx\XKV@prefix\empty\else
    \edef\XKV@prefix{\XKV@prefix @}%
  \fi
}
```

`\XKV@makehd` Creates the header, like `prefix@family@` in `\prefix@family@key`. If `family` is empty, the header reduces to `prefix@`.

```
\def\XKV@makehd#1{%
  \XKV@@zs@def\XKV@resa{#1}%
  \ifx\XKV@resa\empty
    \edef\XKV@header{\XKV@prefix}%
  \else
    \edef\XKV@header{\XKV@prefix\XKV@resa @}%
  \fi
}
```

`\XKV@testopta` `\XKV@t@stopta` `\XKV@t@st@pta` `\XKV@@t@st@pta` Macros for `\setkeys` and `\setrmkeys` for testing for optional arguments and inserting default values. These also perform some necessary initializations.

```
\def\XKV@testopta#1{%
  \XKV@ifstar{\XKV@sttrue\XKV@t@stopta#1}{\XKV@stfalse\XKV@t@stopta#1}%
}
\def\XKV@t@stopta#1{%
  \XKV@ifplus{\XKV@pltrue\XKV@t@st@pta#1}{\XKV@plfalse\XKV@t@st@pta#1}%
}
\def\XKV@t@st@pta#1{\@ifnextchar[{\XKV@@t@st@pta#1}{\XKV@@t@st@pta#1[KV]}}
\def\XKV@@t@st@pta#1[#2]#3{%
  \XKV@makepf{#2}%
  \XKV@@zs@def\XKV@fams{#3}%
  \@ifnextchar[#1{#1[]}%
}
```

`\define@key` `\XKV@define@key` Macro to define a key in a family. Original but modified keyval code. Notice the use of the `KV` prefix as default prefix. This is done to allow setting both keyval and xkeyval keys with a single command.

```
\def\define@key{\XKV@ifstar{\XKV@sttrue\XKV@define@key}{\XKV@stfalse\XKV@define@key}}
\def\XKV@define@key{\@ifnextchar[\XKV@d@fine@key{\XKV@d@fine@key[KV]}}
```

`\XKV@d@fine@key` Workhorse for `\define@key`.

```
\def\XKV@d@fine@key[#1]#2#3{%
```

Set prefix.

```
  \XKV@makepf{#1}%
```

Set header.

```
  \XKV@makehd{#2}%
```

Check whether key macro already exists.

```
  \ifXKV@st
    \expandafter\ifx\csname\XKV@header#3\endcsname\relax\else
      \XKV@err{redefining existing macro `\@backslashchar\XKV@header#3'}%
    \fi
  \fi
```

Define the key macro.

```
  \@ifnextchar[{\XKV@d@fine@k@y{#3}}{%
    \expandafter\def\csname\XKV@header#3\endcsname####1}%
}
```

`\XKV@d@fine@k@y` Defines a key macro and a default value macro.

```
\def\XKV@d@fine@k@y#1[#2]{%
  \expandafter\def\csname\XKV@header#1@default\expandafter\endcsname
    \expandafter{\csname\XKV@header#1\endcsname{#2}}%
  \expandafter\def\csname\XKV@header#1\endcsname##1%
}
```

`\setkeys` Set keys. The stared version does not produce errors, but appends keys that cannot be located to the list in `\XKV@rm`. The plus version sets keys in all families that are supplied. Use `\XKV@testopta` to handle optional arguments.

```
\def\setkeys{\XKV@testopta\XKV@setkeys}
```

`\XKV@setkeys` Workhorse for `\setkeys`. It starts the loop over the options list.

```
\def\XKV@setkeys[#1]#2{%
  \XKV@@zs@def\XKV@keysnot{#1}%
  \let\XKV@rm\empty
```

Sanitize the input and start the loop over keys. We do not use `\@for` since that expands fragile macros. Instead we adopt the methodology of the keyval package.

```
  \def\XKV@tempa{#2}%
  \@selective@sanitize^\XKV@tempa
  \expandafter\XKV@s@tkeys\XKV@tempa,\@nil,%
}
```

`\XKV@s@tkeys` Workhorse for `\XKV@setkeys`.

```
\def\XKV@s@tkeys#1,{%
  \ifx\@nil#1\empty\else
    \XKV@knftrue
```

Split key and value.

```
    \XKV@split#1==\@nil
```

Check whether the key has been found.

```
    \ifXKV@knf
      \ifx\XKV@inpox\@undefined
```

If not in the options section, raise an error or add the key to the list in `\XK@rm` when `\setkeys*` has been used.

```
        \ifXKV@st
```

```
158           \ifx\XKV@rm\empty
159             \toks@{#1}%
160             \xdef\XKV@rm{\the\toks@}%
161           \else
162             \toks@\expandafter{\XKV@rm,#1}%
163             \xdef\XKV@rm{\the\toks@}%
164           \fi
165         \else
166           \XKV@err{‘\XKV@tkey’ undefined in families ‘\XKV@fams’}%
167         \fi
168       \else
```

We are in the options section. Try to use the macro defined by `\DeclareOptionX*`.

```
169         \ifx\XKV@doxs\relax
```

For classes, ignore unknown (possibly global) options. For packages, raise the standard LaTeX error.

```
170           \ifx\@currext\@clsextension\else
171             \let\CurrentOption\XKV@tkey
172             \@unknownoptionerror
173           \fi
174         \else
```

Pass the option through `\DeclareOptionX*`.

```
175           \def\CurrentOption{#1}%
176           \XKV@doxs
```

Remove the option from `\@unusedoptionlist`.

```
177           \XKV@useoption\CurrentOption
178         \fi
179       \fi
180     \else
```

Remove global options set by the document class from `\@unusedoptionlist`. Global options set by other packages or class will be removed by `\ProcessOptionsX*`.

```
181       \ifx\XKV@inpox\@undefined\else\ifx\XKV@testclass\XKV@documentclass
182         \XKV@useoption{#1}%
183       \fi\fi
184     \fi
185     \expandafter\XKV@s@tkeys
186   \fi
187 }
```

`\XKV@split` Macro that splits keys and values.

```
188 \def\XKV@split#1=#2=#3\@nil{%
```

Remove spaces from key input.

```
189   \XKV@@zs@def\XKV@tkey{#1}%
```

If the key is empty and a value has been specified, generate an error.

```
190   \ifx\XKV@tkey\empty
191     \ifx\empty#2\empty\else
```

```
      \toks@{#2}%
      \XKV@err{No key specified for value `\the\toks@'}%
    \fi
    \XKV@knffalse
  \else
```

If in the `\XKV@keysnot` list, ignore the key.

```
    \in@{\expandafter,\XKV@tkey,}{\expandafter,\XKV@keysnot,}%
    \ifin@\XKV@knffalse\else
      \KV@@sp@def\XKV@tempa{#2}%
      \toks@\expandafter{\XKV@tempa}%
```

Save key values if requested.

```
      \ifrecordkeyvals
        \expandafter\xdef\csname XKV@\XKV@tkey @value\endcsname{\the\toks@}%
      \fi
```

Replace pointers by saved value and start the loop over families.

```
      \expandafter\XKV@replacepointers\the\toks@^{}\@nil
      \expandafter\XKV@lf@setkey@infam\the\toks@\@nil{#3}%
    \fi
  \fi
}
```

`\XKV@replacepointers` Replaces all pointers safely by their saved value. The result is stored in `\toks@`. We feed that every time to the macro itself to replace nested pointers. It stops when no pointers are found anymore.

```
\def\XKV@replacepointers#1^#2#3\@nil{%
  \toks@{#1}%
  \ifx\empty#2\empty\else
    \expandafter\let\expandafter\XKV@resa\csname XKV@#2@value\endcsname
    \ifx\XKV@resa\relax
      \XKV@err{No value recorded for key `#2'}%
      \XKV@knffalse
    \else
      \edef\XKV@resb{\the\toks@}%
      \toks@\expandafter\expandafter\expandafter
        {\expandafter\XKV@resb\XKV@resa#3}%
      \expandafter\XKV@replacepointers\the\toks@\@nil
    \fi
  \fi
}
```

`\XKV@lf@setkey@infam` Contains the loops over families. The kind of loop depends on the use of `+` in the command.

```
\def\XKV@lf@setkey@infam#1\@nil#2{%
  \ifx\XKV@fams\empty
```

If the family list is empty, only check the empty family for keys.

```
    \XKV@makehd{}%
    \XKV@setkey@infam{#1}{#2}%
  \else
```

If a command with a `+` is used, set keys in all families on the list. Since there is no danger of expanding fragile macros, we use `\@for`.

```
    \ifXKV@pl
      \@for\XKV@tfam:=\XKV@fams\do{%
        \XKV@makehd\XKV@tfam
        \XKV@setkey@infam{#1}{#2}%
      }%
    \else
```

Else, scan the families on the list but stop when the key is found or when the list has run out.

```
      \XKV@whilist\XKV@tfam:=\XKV@fams\do\ifXKV@knf\fi{%
        \XKV@makehd\XKV@tfam
        \XKV@setkey@infam{#1}{#2}%
      }%
    \fi
  \fi
}
```

`\XKV@setkey@infam` Sets a key in a family. Based on keyval code.

```
\def\XKV@setkey@infam#1#2{%
  \expandafter\let\expandafter\XKV@tempa
    \csname\XKV@header\XKV@tkey\endcsname
```

Check whether the key macro is defined.

```
  \ifx\XKV@tempa\relax\else
    \XKV@knffalse
    \ifx\empty#2\empty
```

No value given, use default.

```
      \expandafter\let\expandafter\XKV@tempb
        \csname\XKV@header\XKV@tkey @default\endcsname
      \ifx\XKV@tempb\relax
        \XKV@err{No value specified for key `\XKV@tkey'}%
      \else
```

Execute key with the default value.

```
        \expandafter\XKV@default\XKV@tempb\@nil
      \fi
    \else
```

Execute key with submitted value.

```
      \XKV@tempa{#1}\relax
    \fi
  \fi
}
```

`\XKV@default` This macro checks the `\prefix@fam@key@default` macro. If the macro has the form as defined by keyval or xkeyval, it is possible to extract the default value and safe that (if requested) and replace pointers. If the form is incorrect, just execute the macro and forget about possible pointers. The reason for this check is that

certain packages (like fancyvrb) abuse the 'default value system' to execute code instead of setting keys by redefining default value macros. These macros do not actually contain a default value and trying to extract that would not work.

```
\def\XKV@default#1#2\@nil{%
```

Retrieve the name of the first token in the macro.

```
  \expandafter\edef\expandafter\XKV@resa\expandafter{\expandafter\@gobble\string#1}%
```

Construct the name that we expect on the basis of the keyval and xkeyval syntax of default values.

```
  \edef\XKV@resb{\XKV@header\XKV@tkey}%
```

Sanitize `\XKV@resb` to reset catcodes for comparison with `\XKV@resa`.

```
  \@onelevel@sanitize\XKV@resb
  \ifx\XKV@resa\XKV@resb
```

If it is safe, extract the value. We temporarily redefine the key macro to save the default value in a token. Saving the default value itself directly to a macro would of course be easier, but a lot of packages rely on this system created by keyval, so we have to support it here.

```
    \begingroup
      \expandafter\def\csname\XKV@header\XKV@tkey\endcsname##1{%
        \global\toks@{##1}%
      }%
      \XKV@tempb
    \endgroup
```

Save the default value to a macro and sanitize the input.

```
    \edef\XKV@tempb{\the\toks@}%
    \@selective@sanitize^\XKV@tempb
```

Safe the default value to a value macro if requested.

```
    \ifrecordkeyvals
      \expandafter\expandafter\expandafter\gdef
        \expandafter\expandafter\csname XKV@\XKV@tkey @value\endcsname
        \expandafter{\XKV@tempb}%
    \fi
```

Replace the pointers.

```
    \expandafter\XKV@replacepointers\XKV@tempb^{}\@nil
```

Execute the default value macro with the (possibly changed) default value.

```
    \expandafter\XKV@tempa\expandafter{\the\toks@}%
  \else
```

Execute the default value macro without any features.

```
    \XKV@tempb\relax
  \fi
}
```

`\setrmkeys` Set remaining keys stored in `\XKV@rm`. The stared version creates a new list in `\XKV@rm` in case there are still keys that cannot be located in the families specified.

Care is taken again not to expand fragile macros. Use `\XKV@testopa` again to handle optional arguments.

```
\def\setrmkeys{\XKV@testopta\XKV@setrmkeys}
```

`\XKV@setrmkeys` Submits the keys in `\XKV@rm` to `\XKV@setkeys`.

```
\def\XKV@setrmkeys[#1]{%
  \toks@\expandafter{\XKV@rm}%
  \edef\XKV@tempa{\noexpand\XKV@setkeys[#1]{\the\toks@}}%
  \XKV@tempa
}
```

`\XKV@ifku` This macro allows checking if a key is defined in a family from a list of families.

```
\def\XKV@ifku{\@ifnextchar[\XKV@ifk@{\XKV@ifk@[KV]}}
```

`\XKV@ifk@` Workhorse for `\XKV@ifku`.

```
\long\def\XKV@ifk@[#1]#2#3#4#5{%
```

Set the prefix.

```
  \XKV@knftrue
  \XKV@makepf{#1}%
  \XKV@@zs@def\XKV@fams{#2}%
```

Treat the case that the list of families is empty independently.

```
  \ifx\XKV@fams\empty
```

Set the header.

```
    \XKV@makehd{}%
```

Check whether the macro for the key is defined.

```
    \expandafter\ifx\csname\XKV@header#3\endcsname\relax\else
      \XKV@knffalse
    \fi
  \else
```

Loop over possible families.

```
    \XKV@whilist\XKV@tfam:=\XKV@fams\do\ifXKV@knf\fi{%
```

Set the header.

```
      \XKV@makehd\XKV@tfam
```

Check whether the macro for the key is defined.

```
      \expandafter\ifx\csname\XKV@header#3\endcsname\relax\else
        \XKV@knffalse
      \fi
    }%
  \fi
```

Execute one of the final two arguments depending on state of `\XKV@knf`.

```
  \ifXKV@knf#4\else#5\fi
}
```

Finalize.

```
\catcode`\@=\XKVAtCode\relax
\catcode`\^=\XKVHatCode\relax
⟨/tex⟩
```

13.2 LaTeX program

Initialize the package.

```
%<*latex>
\NeedsTeXFormat{LaTeX2e}[1995/12/01]
\ProvidesPackage{xkeyval}[2004/08/24 v1.4 key=value parser (HA)]
```

Initializations. Load `xkeyval.tex`. Avoid loading keyval later on.

```
\def\XKeyValLoaded{}
\input{xkeyval.tex}
\let\XKV@doxs\relax
\@namedef{ver@keyval.sty}{1999/03/16}
```

`\XKV@err` Error macro.

```
\def\XKV@err#1{\PackageError{xkeyval}{#1}\@ehc}
```

`\XKV@testoptb` `\XKV@t@stoptb` `\XKV@t@st@ptb` Macros for `\ExecuteOptionsX` and `\ProcessOptionsX` for testing for optional arguments and inserting default values.

```
\def\XKV@testoptb#1{\@testopt{\XKV@t@stoptb#1}{KV}}
\def\XKV@t@stoptb#1[#2]{%
  \@ifnextchar<{\XKV@t@st@ptb#1[#2]}{\XKV@t@st@ptb#1[#2]<\@currname>}%
}
\def\XKV@t@st@ptb#1[#2]<#3>{%
  \@ifnextchar[{#1[#2]{#3}}{#1[#2]{#3}[]}%
}
```

`\XKV@getdocumentclass` Retrieve the document class from `\@filelist`. This is the first filename in the list with a class extension. Use a while loop to scan the list and stop when we found the first filename which is a class. Also stop in case the list is scanned fully.

```
\def\XKV@getdocumentclass{%
  \XKV@whilist\XKV@tempa:=\@filelist\do\ifx\XKV@documentclass\@undefined\fi{%
    \filename@parse\XKV@tempa
    \ifx\filename@ext\@clsextension
      \edef\XKV@documentclass{\filename@base.\filename@ext}%
    \fi
  }
}
```

`\XKV@filterclassoptions` Code to filter `key=value` pairs from `\@classoptionslist`. Notice that no attempt is being made to protect fragile macros.

```
\def\XKV@filterclassoptions{%
  \let\XKV@tempa\empty
  \@for\XKV@tempb:=\@classoptionslist\do{%
    \ifx\XKV@tempb\empty\else
      \@expandtwoargs\in@{=}{\XKV@tempb}%
      \ifin@\else
        \edef\XKV@tempa{%
          \XKV@tempa
          \ifx\XKV@tempa\empty\else,\fi
          \XKV@tempb
```

```
        }%
      \fi
    \fi
  }%
  \let\@classoptionslist\XKV@tempa
}
```

At loading, retrieve document class, copy `\@classoptionslist` to `\XKV@classoptionslist` and filter `key=value` pairs from the original.

```
\ifx\XKV@documentclass\@undefined
  \XKV@getdocumentclass
  \ifx\XKV@documentclass\@undefined
    \XKV@err{xkeyval loaded before \@backslashchar documentclass}%
    \let\XKV@documentclass\empty
    \let\XKV@classoptionslist\empty
  \else
    \let\XKV@classoptionslist\@classoptionslist
    \XKV@filterclassoptions
  \fi
\fi
```

`\XKV@getoption` Retrieves `option` from `option=value`.

```
\def\XKV@getoption#1=#2\@nil{\def\CurrentOption{#1}}
```

`\XKV@useoption` Removes an option from `\@unusedoptionlist`.

```
\def\XKV@useoption#1{%
  \@expandtwoargs\@removeelement{#1}\@unusedoptionlist\@unusedoptionlist
}
```

Macros for class and package writers. These are mainly shortcuts to `\define@key` and `\setkeys`. The LaTeX macro `\@fileswith@pti@ns` is set to generate an error. This is the case when a class or package is loaded in between `\DeclareOptionX` and `\ProcessOptionsX` commands.

`\DeclareOptionX` Declare an option.

```
\def\DeclareOptionX{%
  \let\@fileswith@pti@ns\@badrequireerror
  \XKV@ifstar\XKV@dox\XKV@d@x
}
```

`\XKV@dox` This macro defines `\XKV@doxs` to be used for unknown options.

```
\long\def\XKV@dox#1{\toks@{#1}\edef\XKV@doxs{\the\toks@}}
```

`\XKV@d@x` `\XKV@@d@x` `\XKV@@@d@x` Insert default prefix and family name (which is the filename of the class or package) and add empty default value if none present. Execute `\define@key`.

```
\def\XKV@d@x{\@testopt\XKV@@d@x{KV}}
\def\XKV@@d@x[#1]{\@ifnextchar<{\XKV@@@d@x[#1]}{\XKV@@@d@x[#1]<\@currname>}}
\def\XKV@@@d@x[#1]<#2>#3{\@testopt{\define@key[#1]{#2}{#3}}{}}
```

`\ExecuteOptionsX` This macro sets keys to specified values and uses `\setkeys` to do the job. Insert

default prefix and family name if none provided. Use `\XKV@testoptb` to handle optional arguments.

```
\def\ExecuteOptionsX{\XKV@testoptb\setkeys}
```

`\ProcessOptionsX` `\XKV@pox` Processes class or package using xkeyval. The stared version copies class options submitted by the user as well, given that they are defined in the local families which are passed to the macro. Use `\XKV@testoptb` again to handle optional arguments.

```
\def\ProcessOptionsX{\XKV@ifstar{\XKV@sttrue\XKV@pox}{\XKV@stfalse\XKV@pox}}
\def\XKV@pox{\XKV@testoptb\XKV@p@x}
```

`\XKV@p@x` Workhorse for `\ProcessOptionsX` and `\ProcessOptionsX*`.

```
\def\XKV@p@x[#1]#2[#3]{%
  \let\XKV@tempa\empty
```

Set `\XKV@inpox`: indicates that we are in `\ProcessOptionsX` to invoke a special routine in `\XKV@s@tkeys`.

```
  \let\XKV@inpox\empty
```

Set `\@fileswith@pti@ns` again in case no `\DeclareOptionX` has been used. This will be used to identify a call to `\setkeys` from `\ProcessOptionsX`.

```
  \let\@fileswith@pti@ns\@badrequireerror
  \edef\XKV@testclass{\@currname.\@currext}%
```

If xkeyval is loaded by the document class, initialize `\@unusedoptionlist`.

```
  \ifx\XKV@testclass\XKV@documentclass
    \let\@unusedoptionlist\XKV@classoptionslist
  \else
```

Else, if the stared version is used, copy global options in case they are defined in local families. Do not execute this in the document class to avoid setting keys twice.

```
    \ifXKV@st
      \@for\XKV@tempb:=\XKV@classoptionslist\do{%
        \expandafter\XKV@getoption\XKV@tempb=\@nil
        \XKV@ifku[#1]{#2}{\CurrentOption}{\relax}{%
```

If the option also exists in local families, add it to the list for later use and remove it from `\@unusedoptionlist`.

```
          \XKV@useoption\XKV@tempb
          \edef\XKV@tempa{\XKV@tempa\XKV@tempb,}%
        }%
      }%
    \fi
  \fi
```

Set options. We can be certain that global options can be set since the definitions of local options have been checked above. Note that `\DeclareOptionX*` will not consume global options when `\ProcessOptionsX*` is used.

```
  \edef\XKV@tempb{%
    \noexpand\setkeys[#1]{#2}[#3]{\XKV@tempa\@ptionlist{\@currname.\@currext}}%
```

```
  }%
  \XKV@tempb
```

Reset the macro created by `\DeclareOptionX*` to avoid processing future unknown keys using `\XKV@doxs`.

```
  \let\XKV@doxs\relax
```

Reset the `\XKV@rm` macro to avoid processing remaining options with `\setrmkeys`.

```
  \let\XKV@rm\empty
```

Reset `\XKV@inpox`: not in `\ProcessOptionsX` anymore.

```
  \let\XKV@inpox\@undefined
```

Reset `\@fileswith@pti@ns` to allow loading of classes or packages again.

```
  \let\@fileswith@pti@ns\@@fileswith@pti@ns
  \AtEndOfPackage{\let\@unprocessedoptions\relax}%
}
```

The options section. Postponed to the end to allow for using xkeyval options macros. All options are silently ignored.

```
\DeclareOptionX*{\PackageWarning{xkeyval}{Unknown option `\CurrentOption'}}
\ProcessOptionsX
```

⟨/latex⟩

13.3 keyval primitives

Since the xkeyval macros handle input in a very different way than keyval macros, it is not wise to redefine keyval primitives (like `\KV@do` and `\KV@split`) used by other packages as a back door into `\setkeys`. Instead, we load the original primitives here for compatibility to existing packages using (parts of) keyval. Most of the code is original, but slightly adapted to xkeyval. See the keyval documentation for information about the macros below.

```
%<*keyval>
%%
%% Taken from keyval.sty.
%%
\def\XKV@tempa#1{%
\def\KV@@sp@def##1##2{%
  \futurelet\XKV@resa\KV@@sp@d##2\@nil\@nil#1\@nil\relax##1}%
\def\KV@@sp@d{%
  \ifx\XKV@resa\@sptoken
    \expandafter\KV@@sp@b
  \else
    \expandafter\KV@@sp@b\expandafter#1%
  \fi}%
\def\KV@@sp@b#1##1 \@nil{\KV@@sp@c##1}%
  }
\XKV@tempa{ }
\def\KV@@sp@c#1\@nil#2\relax#3{\toks@{#1}\edef#3{\the\toks@}}
\def\KV@do#1,{%
```

```
\ifx\relax#1\empty\else
 \KV@split#1==\relax
 \expandafter\KV@do\fi}
\def\KV@split#1=#2=#3\relax{%
 \KV@@sp@def\XKV@tempa{#1}%
 \ifx\XKV@tempa\empty\else
   \expandafter\let\expandafter\XKV@tempc
     \csname\KV@prefix\XKV@tempa\endcsname
   \ifx\XKV@tempc\relax
     \XKV@err{‘\XKV@tempa’ undefined}%
   \else
     \ifx\empty#3\empty
       \KV@default
     \else
       \KV@@sp@def\XKV@tempb{#2}%
       \expandafter\XKV@tempc\expandafter{\XKV@tempb}\relax
     \fi
   \fi
 \fi}
\def\KV@default{%
 \expandafter\let\expandafter\XKV@tempb
   \csname\KV@prefix\XKV@tempa @default\endcsname
 \ifx\XKV@tempb\relax
   \XKV@err{No value specified for key ‘\XKV@tempa’}%
 \else
   \XKV@tempb\relax
 \fi}
```
⟨/keyval⟩

13.4 TEX header

This section generates `xkeyval.def` which contains some standard LaTeX macros taken from `latex.ltx`. This will only be loaded when not using `xkeyval.sty`.

```
%<*header>
%%
%% Taken from latex.ltx.
%%
\def\@nnil{\@nil}
\newtoks\@temptokena
\long\def\@firstoftwo#1#2{#1}
\long\def\@secondoftwo#1#2{#2}
\long\def\@ifnextchar#1#2#3{%
 \let\reserved@d=#1%
 \def\reserved@a{#2}%
 \def\reserved@b{#3}%
 \futurelet\@let@token\@ifnch}
\def\@ifnch{%
 \ifx\@let@token\@sptoken
   \let\reserved@c\@xifnch
```

```
  \else
    \ifx\@let@token\reserved@d
      \let\reserved@c\reserved@a
    \else
      \let\reserved@c\reserved@b
    \fi
  \fi
  \reserved@c}
\def\:{\let\@sptoken= } \:
\def\:{\@xifnch} \expandafter\def\: {\futurelet\@let@token\@ifnch}
\def\@fornoop#1\@@#2#3{}
\long\def\@for#1:=#2\do#3{%
  \expandafter\def\expandafter\@fortmp\expandafter{#2}%
  \ifx\@fortmp\empty \else
    \expandafter\@forloop#2,\@nil,\@nil\@@#1{#3}\fi}
\long\def\@forloop#1,#2,#3\@@#4#5{\def#4{#1}\ifx #4\@nnil \else
       #5\def#4{#2}\ifx #4\@nnil \else#5\@iforloop #3\@@#4{#5}\fi\fi}
\long\def\@iforloop#1,#2\@@#3#4{\def#3{#1}\ifx #3\@nnil
       \expandafter\@fornoop \else
      #4\relax\expandafter\@iforloop\fi#2\@@#3{#4}}
\long\def \@gobble #1{}
\edef\@backslashchar{\expandafter\@gobble\string\\}
\newif\ifin@
\def\in@#1#2{%
 \def\in@@##1#1##2##3\in@@{%
  \ifx\in@##2\in@false\else\in@true\fi}%
 \in@@#2#1\in@\in@@}
\def\zap@space#1 #2{%
  #1%
  \ifx#2\empty\else\expandafter\zap@space\fi
  #2}
\def\strip@prefix#1>{}
\def \@onelevel@sanitize #1{%
  \edef #1{\expandafter\strip@prefix
            \meaning #1}%
}
```
⟨/header⟩

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.

Symbols

\: 479, 480
\@ 2, 4, 310
\@@ . . 82, 85, 96, 98, 481, 485–488, 490
\@@cmd 46, 50, 52, 54, 66, 72, 74
\@@fileswith@pti@ns 403
\@@selective@sanitize 39, 41
\@@tok 42, 44, 45, 48, 50, 55, 58, 71, 72

\@backslashchar 55, 132, 355, 492
\@badrequireerror 368, 381
\@classoptionslist 338, 350, 359
\@clsextension 170, 331
\@currext . 170, 382, 397
\@currname 323, 373, 382, 397
\@ehc 320
\@empty 75
\@expandtwoargs 340, 365
\@filelist 329
\@fileswith@pti@ns 368, 381, 403
\@firstoftwo 28, 77, 78, 461
\@for . 230, 338, 387, 482
\@forloop 485, 486
\@fornoop 481, 489
\@fortmp 483, 484
\@gobble 46, 261, 491, 492
\@i . . . 42, 50, 66, 72, 75
\@ifnch . . . 467, 468, 480
\@ifncharacter 21
\@ifnextchar 22, 23, 119, 123, 126, 135, 290, 323, 326, 373, 463
\@ifnextcharacter 21, 77, 78
\@iforloop 487, 488, 490
\@ii 42, 43
\@iii 44, 55
\@iv 52, 56
\@let@token 467, 469, 472, 480
\@namedef 319
\@nil 82, 149, 152, 154, 188, 204, 205, 209, 220, 224, 253, 260, 278, 363, 388, 415, 422, 425, 459, 485
\@nnil . 87, 459, 486–488
\@onelevel@sanitize 37, 263, 503
\@ptionlist 397
\@removeelement . . . 365
\@secondoftwo . . 30, 462
\@selective@sanitize 34, 148, 272
\@sptoken 48, 71, 72, 417, 469, 479
\@temptokena 57, 62, 63, 65, 68, 71, 72, 460
\@testopt . 321, 372, 374
\@undefined 6, 156, 181, 329, 352, 354, 402
\@unknownoptionerror 172
\@unprocessedoptions 404
\@unusedoptionlist 365, 384
\@xifnch 470, 480
\\ 492
\^ 3, 5, 311

A

\afterassignment . . 50
\AtEndOfPackage . . . 404

B

\begingroup . 35, 59, 265
\bgroup 22, 58

C

\catcode . . 2–5, 310, 311
\CurrentOption 171, 175, 177, 363, 389, 406

D

\DeclareOptionX *26*, 367, 406
\define@key . . . 125, 374

E

\endgroup . . 40, 65, 270
\errmessage 76
\ExecuteOptionsX *26*, 375

F

\filename@base 332
\filename@ext . 331, 332
\filename@parse . . . 330

G

\gdef 274
\global 267

I

\if 27, 55
\ifin@ . 70, 198, 341, 493
\ifreckeyvals *15*
\ifrecordkeyvals 20, 201, 273
\ifXKV@knf 18, 155, 235, 301, 308
\ifXKV@pl 17, 229
\ifXKV@st 16, 130, 157, 386
\in@ 69, 197, 340, 494, 496, 497
\in@@ 495, 497
\in@false 496
\in@true 496
\input 8, 11, 317

K

\KV@@sp@b . 418, 420, 422
\KV@@sp@c 422, 425
\KV@@sp@d 415, 416
\KV@@sp@def 199, 414, 431, 441
\KV@default . . . 439, 446
\KV@do 426, 429
\KV@err 13
\KV@errx 12, 13
\KV@prefix . . . 434, 448
\KV@split 428, 430

M

\meaning 44, 505
\message 7

N

\NeedsTeXFormat . . . 314

P

\PackageError 320
\PackageWarning . . . 406
\ProcessOptionsX 376, 407
\ProvidesPackage . . 315

R

\reserved@a . . . 465, 473

\reserved@b . . . 466, 475
\reserved@c 470, 473, 475, 478
\reserved@d . . . 464, 472

S

\setkeys *19*, 143, 375, 397
\setrmkeys *23*, 284
\strip@prefix . 502, 504

T

\toks@ 36, 38, 49, 57, 65, 70, 159, 160, 162, 163, 192, 193, 200, 202, 204, 205, 210, 217, 218, 220, 267, 271, 279, 286, 287, 371, 425

X

\XKeyValLoaded . . 6, 316
\XKV@@@d@x 372
\XKV@@d@x 372
\XKV@@t@st@pta 113
\XKV@@zs@def *17*, 79, 100, 106, 122, 145, 189, 294
\XKV@classoptionslist . 357, 359, 384, 387
\XKV@d@fine@k@y *19*, 135, 138
\XKV@d@fine@key 126, 127
\XKV@d@x 369, 372
\XKV@default . . 253, 260
\XKV@define@key . . . 125
\XKV@documentclass . . 181, 329, 332, 352, 354, 356, 383
\XKV@dox . . *26*, 369, 371
\XKV@doxs 169, 176, 318, 371, 400
\XKV@err 12, *16*, *25*, 76, 132, 166, 193, 214, 251, 320, 355, 436, 450
\XKV@fams 122, 166, 225, 230, 235, 294, 295, 301
\XKV@filterclassoptions 336, 360
\XKV@getdocumentclass *25*, 328, 353
\XKV@getoption *26*, 363, 388
\XKV@header 108, 110, 131, 132, 136, 139–141, 244, 249, 262, 266, 297, 303
\XKV@ifk@ . *24*, 290, 291
\XKV@ifku 290, 389
\XKV@ifplus . *17*, 78, 117
\XKV@ifstar . . *17*, 77, 114, 125, 369, 376
\XKV@inpox 156, 181, 380, 402
\XKV@keysnot . . 145, 197
\XKV@knffalse 195, 198, 215, 246, 298, 304
\XKV@knftrue . . 153, 292
\XKV@lf@setkey@infam 205, 224
\XKV@makehd *18*, 105, 129, 226, 231, 236, 296, 302
\XKV@makepf *18*, 99, 121, 128, 293
\XKV@p@x . . *27*, 377, 378
\XKV@plfalse 117
\XKV@pltrue 117
\XKV@pox 376
\XKV@prefix 100–102, 108, 110
\XKV@replacepointers *21*, 204, 209, 220, 278
\XKV@resa 106, 107, 110, 212, 213, 219, 261, 264, 415, 417
\XKV@resb 217, 219, 262–264
\XKV@rm 19, 146, 158, 160, 162, 163, 286, 401
\XKV@s@tkeys . . 149, 151
\XKV@setkey@infam . . 227, 232, 237, 242
\XKV@setkeys 143, 144, 287
\XKV@setrmkeys *24*, 284, 285
\XKV@split . . . 154, 188
\XKV@stfalse 114, 125, 376
\XKV@sttrue 114, 125, 376
\XKV@t@st@pta 113
\XKV@t@st@ptb 321
\XKV@t@stopta 113
\XKV@t@stoptb 321
\XKV@tempa 147–149, 199, 200, 243, 245, 256, 279, 287, 288, 329, 330, 337, 342–344, 350, 379, 391, 397, 413, 424, 431, 432, 434, 436, 448, 450
\XKV@tempb . . . 248, 250, 253, 269, 271, 272, 276, 278, 281, 338–340, 345, 387, 388, 390, 391, 396, 399, 441, 442, 447, 449, 452
\XKV@tempc 433, 435, 442
\XKV@testclass 181, 382, 383
\XKV@testopta 113, 143, 284
\XKV@testoptb 321, 375, 377
\XKV@tfam . 230, 231, 235, 236, 301, 302
\XKV@tkey . 166, 171, 189, 190, 197, 202, 244, 249, 251, 262, 266, 275
\XKV@useoption . *26*, 177, 182, 364, 390
\XKV@wh@l@st *17*, 88, 93, 98
\XKV@wh@list 82, 85
\XKV@whilist 80, 235, 301, 329
\XKVAtCode 2, 310
\XKVHatCode 3, 311

Z

\zap@space 79, 498, 500