The \texttt{xkeyval} package \footnote{This package can be downloaded from the CTAN mirrors: \texttt{/macros/latex/contrib/xkeyval}. See \texttt{xkeyval.dtx} for information on installing \texttt{xkeyval} into your \TeX\ or \LaTeX\ distribution and for the license of this package.}

Hendri Adriaens

v2.7a (2014/12/03)

Abstract

This package is an extension of the \texttt{keyval} package and offers more flexible macros for defining and setting keys. The package provides a pointer and a preset system. Furthermore, it supplies macros to allow class and package options to contain options of the \texttt{key=value} form. A \TeX\ kernel patch is provided to avoid premature expansions of macros in class or package options. A specialized system for setting \texttt{PSTricks} keys is provided by the \texttt{pst-xkey} package.

Contents

1 Introduction \hfill 2
2 Loading \texttt{xkeyval} \hfill 3
3 Defining and managing keys \hfill 3
   3.1 Ordinary keys \hfill 4
   3.2 Command keys \hfill 4
   3.3 Choice keys \hfill 5
   3.4 Boolean keys \hfill 7
   3.5 Checking keys \hfill 8
   3.6 Disabling keys \hfill 8
4 Setting keys \hfill 8
   4.1 The user interface \hfill 8
   4.2 A few details \hfill 10
5 Pointers \hfill 11
   5.1 Saving values \hfill 11
   5.2 Using saved values \hfill 13
6 Presetting keys \hfill 15
7 Package option processing \hfill 17
8 List of macro structures \hfill 19
9 Warnings and errors \hfill 21
10 Category codes \hfill 22
11 Known issues \hfill 23
12 Additional packages \hfill 24
   12.1 \texttt{xkview} \hfill 24
   12.2 \texttt{xkltex} \hfill 25
   12.3 \texttt{pst-xkey} \hfill 26
13 Examples and documentation \hfill 27
14 Implementation \hfill 28
   14.1 \texttt{xkeyval.tex} \hfill 28
   14.2 \texttt{xkeyval.sty} \hfill 47
   14.3 \texttt{keyval.tex} \hfill 50
   14.4 \texttt{xktxhdr.tex} \hfill 52
   14.5 \texttt{xkutils.tex} \hfill 53
   14.6 \texttt{xkview.sty} \hfill 61
   14.7 \texttt{xkltex.sty} \hfill 65
   14.8 \texttt{pst-xkey.tex} \hfill 66
   14.9 \texttt{pst-xkey.sty} \hfill 67
References \hfill 68
Acknowledgements \hfill 68
Version history \hfill 68
Index \hfill 70
1 Introduction

This package is an extension of the keyval package by David Carlisle [3] and offers more flexible and robust macros for defining and setting keys. Using keys in macro definition has the advantage that the 9 arguments maximum can easily be avoided and that it reduces confusion in the syntax of your macro when compared to using a lot of (optional) arguments. Compare for instance the following possible syntaxes of the macro \mybox which might for instance use its arguments to draw some box containing text.

\mybox{8pt}{20pt}{some text}[red][white][blue]
\mybox[text=red,background=white,frame=blue,left=8pt,right=20pt]{some text}

Notice that, to be able to specify the frame color in the first example, the other colors need to be specified as well. This is not necessary in the second example and these colors can get preset values. The same thing holds for the margins.

The idea is that one first defines a set of keys using the tools presented in section 3 in the document preamble or in a package or class. These keys can perform a function with the user input. The way to submit user input to these key macros, is by using one of the user interfaces described in sections 4, 5 and 6. The main user interface is provided by the \setkeys command. Using these interfaces, one can simplify macro syntax and for instance define the \mybox macro above as follows.

\define@key{mybox}[left]{\setlength\myleft{#1}}
\define@key{mybox}[background]{\def\background{#1}}
% and some other keys
\def\mybox\@ifnextchar\def\mybox\@{\mybox\text}
\setkeys{mybox}{left}
% some operations to typset #2

Notice that the combination of the two definitions \mybox and \@mybox can be replaced by \newcommand\mybox[2][]{...} when using \LaTeX.

Both keys defined using the keyval and xkeyval can be set by this package. The xkeyval macros allow for scanning multiple sets (called ‘families’) of keys. This can, for instance, be used to create local families for 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 (pre)set all of the keys from the different families globally.

The package is compatible to plain \TeX and redefines several keyval macros 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 internal keyval macros are supplied in keyval.tex to guarantee compatibility to packages that use those macros. Section 11 provides more information about this issue.

The organization of this documentation is as follows. Section 2 discusses how to load xkeyval and what the package does when it is loaded. Section 3 will discuss the macros available to define and manage keys. Section 4 will continue with describing the macros that can set the keys. Section 5 explains special syntax which will allow saving and copying key values. In section 6, the preset system will be introduced. Section 10 will explain how xkeyval protects itself for catcode changes of the comma and the equality sign by other packages. The xkeyval package also provides commands
to declare and process class and package options that can take values. These will be discussed in section 7. Section 8 provides an overview of structures used to create xkeyval internal macros used for keys, values, presets, etcetera. Sections 9 and 11 discuss feedback that xkeyval might give and known issues, respectively. Section 12 discusses several additional packages that come with the xkeyval bundle. Section 12.1 presents a viewer utility which produces overviews of defined keys. An extension of the \texttt{\LaTeX} kernel with respect to the class and package options system is discussed in section 12.2. This extension provides a way to use expandable macros in package options. Section 12.3 presents the \texttt{psst-xkey} package, which provides an options system based on xkeyval, but which is specialized in setting \texttt{\PSTricks} keys.

Throughout this documentation, you will find some examples with a short description. More examples can be found in the example files that come with this package. See section 13 for more information. This section also provides the information how to generate the full documentation from the source.

## 2 Loading xkeyval

To load the \texttt{xkeyval} package,\footnote{The \texttt{xkeyval} package consists of the files \texttt{xkeyval.tex}, \texttt{xkeyval.sty}, \texttt{xkeyval.tex}, \texttt{xkeyvalx.tex}.} plain \texttt{\LaTeX} users do \texttt{\input xkeyval}. \texttt{\LaTeX} users do one of the following: \texttt{\usepackage{xkeyval}} or \texttt{\RequirePackage{xkeyval}}. The package does not have options. It is mandatory for \texttt{\LaTeX} users to load \texttt{xkeyval} at any point after the \texttt{\documentclass} command. Loading \texttt{xkeyval} from the class which is the document class itself is possible. The package will use the \texttt{\LaTeX}\texttt{e} engine when available. In particular, \texttt{\ifcsname} is used whenever possible to avoid filling \texttt{\LaTeX}'s hash tables with useless entries, for instance when searching for keys in families.

If \texttt{xkeyval} is loaded by \texttt{\RequirePackage} or \texttt{\usepackage}, the package performs two action immediately. These require \texttt{xkeyval} to be loaded at any point after \texttt{\documentclass} or by the document class itself.

First, it retrieves the document class of the document at hand and stores that (including the class extension) into the following macro.

```
\XV@documentclass
```

This macro could, for instance, contain \texttt{article.cls} and can be useful when using \texttt{\ProcessOptionsX} in a class. See page 18.

Secondly, the global options submitted to the \texttt{\documentclass} command and stored by \texttt{\LaTeX} in \texttt{\@classoptionslist} are copied to the following macro.

```
\XV@classoptionslist
```

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

## 3 Defining and managing keys

This section discusses macros to define keys and some tools to manage keys. A useful extension to \texttt{xkeyval} is the \texttt{xkeyview} package. This package defines commands to generate overviews of defined keys. See section 12.1 for more information.
3.1 Ordinary keys

This section describes how to define ordinary keys.

\define@key[\{prefix\}][\{family\}][\{key\}][\{default\}][\{function\}]

This defines a macro of the form \( \texttt{\{prefix\} \{family\} \{key\} \{default\} \{function\} } \). The default value for \( \{prefix\} \) is \texttt{C3/C3}. This is the standard throughout the package to simplify mixing keyval and xkeyval keys. When \( \{key\} \) is used in a \texttt{\setkeys} command (see section 4) containing key=value, the macro \( \texttt{\{prefix\} \{family\} \{key\} } \) receives value as its argument and will be executed. The argument can be accessed by \( \{function\} \) by using \#1 inside the function.

\define@key[\{family\}][\{key\}][\{default\}][\{function\}]

xkeyval will generate an error when the user omits =value for a key in the options list of \texttt{\setkeys} (see section 4). To avoid this, the optional argument can be used to specify a default value.

\define@key[\{family\}][\{key\}][\{default\}][\{function\}]

This will additionally define a macro \( \texttt{\{prefix\} \{family\} \{key\} \{default\} } \) as a macro with no arguments and definition \( \texttt{\{prefix\} \{family\} \{key\} \{default\} } \) which will be used when =value is missing for key in the options list. So, the last example comes down to doing

\define@key[\{family\}][\{key\}][\{default\}][\{function\}]

When \( \{prefix\} \) is specified and empty, the macros created by \texttt{\define@key} will have the form \( \texttt{\{family\} \{key\} } \). When \( \{family\} \) is empty, the resulting form will be \( \texttt{\{prefix\} \{key\} } \). When both \( \{prefix\} \) and \( \{family\} \) are empty, the form is \( \texttt{\{key\} } \). This combination of prefix and family will be called the header. The rules to create the header will be applied to all commands taking (optional) prefix and family arguments.

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.\(^2\)

We now define some keys to be used in examples throughout this documentation.

\define@key[\{family\}][\{key\}][\{default\}][\{function\}]

\define@key[\{family\}][\{key\}][\{default\}][\{function\}]

\define@key[\{family\}][\{key\}][\{default\}][\{function\}]

3.2 Command keys

Command keys are specialized keys that, before executing any code, save the user input to a macro.

\(^2\)Like PStricks, which uses a system originating from keyval, but which has been modified to use no families and \texttt{\setkeys} as prefix.
\define@cmdkey{\prefix}{\family}{\key}{\default}{\function}

\define@cmdkey This has the effect of defining a key macro of the form \(\prefix\family\key\) that, when used, first saves the user input to a macro of the form \(\prefix\key\) and then executes \(\function\). \(\prefix\) is the macro prefix. If \(\prefix\) is not specified, the usual combination of \(\prefix\) and \(\family\), together with the extra prefix \texttt{cmd}, will be used to create the macro prefix, namely \texttt{cmd\prefix\family\key}. The two keys in the following example hence do exactly the same thing.\footnote{Remember that some rules are applied when creating the header, the combination of \(\prefix\) and \(\header\). See section 3.1.}

\define@cmdkey{\prefix\family\key}{none}{value: \texttt{\cmd\prefix\family\key}}
\define@cmdkey{\prefix\family\key}{one}{\(\def\cmd\prefix\family\key\#1\) value: \texttt{\cmd\prefix\family\key}}

The value \texttt{none} is again the default value that will be submitted to the key macro when the user didn’t supply a value. (See also section 3.1 for more information.)

The following two lines also implement a key with the same key macro.

\define@cmdkeys{\prefix}{\family}{\prefix\family}{\prefix\family}{\default}

\define@cmdkeys This repeatedly calls (an internal of) \texttt{\define@cmdkey} for all keys in the list of \(\keys\). Note that it is not possible to specify a custom key function for the keys created by this command. The only function of those keys is to save user input in a macro. The first line and the last two lines of the following example create keys with the same internal key macro.

\define@cmdkeys{\prefix\family\key}{none}{\texttt{\my\key}}
\define@cmdkeys{\prefix\family\key}{one}{\(\def\my\key\#1\) value: \texttt{\my\key}}

Note that the key macro itself in the examples above is still \texttt{\KV\family\key}, just as in the previous example.

A lot of packages define keys that only save their value to a macro so that it can be used later. Using the macro above, one can save some tokens in the package. Some more tokens can be saved by using the following macro.

\define@cmdkeys{\prefix\family\key}{none}{value: \texttt{\my\key}}
\define@cmdkeys{\prefix\family\key}{one}{\(\def\my\key\#1\) value: \texttt{\my\key}}

3.3 Choice keys

Choice keys allow only a limited number of different values for user input. These keys are defined as follows.

\define@choicekey{\prefix\family\key}{\bin}{\(\prefix\family\key\)}{\(\prefix\family\key\)}{\default}{\(\function\)}
\define@choicekey*{\prefix\family\key}{\bin}{\(\prefix\family\key\)}{\(\prefix\family\key\)}{\default}{\(\function\)}

\define@choicekey The keys work the same as ordinary keys, except that, before executing anything, it is verified whether the user input \(\#1\) is present in the comma separated list \(\bin\). The starred version first converts the input in \(\#1\) and \(\bin\) to lowercase before performing the check. If the input is not allowed, an error is produced and the key macro \(\function\).

\footnote{Notice however, that the first key will be listed as a ‘command key’ by \texttt{xkview} and the second as an ‘ordinary key’. See section 12.1.}
will not be executed. If the input is allowed, the key macro \(\langle\text{func}\rangle\) will be executed. \(\langle\text{dft}\rangle\) is submitted to the key macro when the user didn’t supply a value for the key. (See also section 3.1.)

The optional \(\langle\text{bin}\rangle\) should contain either one or two control sequences (macros). The first one will be used to store the user input used in the input check (hence, in lowercase when the starred version was used). The original user input will always be available in \(/AZ/BD\). The second (if present) will contain the number of the input in the \(\langle\text{al}\rangle\) list, starting from 0. The number will be set to \(/B9/BD\) if the input was not allowed. The number can, for instance, be used in a \(\text{\textbackslash ifcase}\) statement in \(\langle\text{func}\rangle\).

These macros operate as their counterparts without the \(/B7\), but allow for specifying two key macros. \(\langle\text{f1}\rangle\) will be executed when the input was correct and \(\langle\text{f2}\rangle\) will be executed when the input was not allowed. Again, the starred version executes the check after converting user input and \(\langle\text{al}\rangle\) to lowercase.

Choice keys work by adding (an internal version\(^5\)) of the \(\text{\textbackslash XV0cc}\) macro to key macros. This macro has similar arguments as the \(\text{\textbackslash define\textunderscore choicekey}\) macro and

\(^5\)See section 14 for details of the implementation of choice keys.
the optional * and + have the same meaning. ⟨input⟩ holds the input that should be checked, namely, whether it is (in lowercase if * was used) in the list ⟨al⟩. One can use this macro to create custom choice keys. See an example below.

\define@key{fam}{key}{% 
  I will first check your input, please wait.\
  #!\if\check{\code{\val}}{true, false}{\% 
    The input \val was correct, we proceed.\% 
  }\{ 
    The input \val was incorrect and was ignored.\% 
  }\{ 
    I finished the input check. 
  } 
}

Try to find out why this key cannot be defined with \define@boolkey which is introduced in the next section.

### 3.4 Boolean keys

This section describes boolean keys which can be either true or false. A boolean key is a special version of a choice key (see section 3.3), where ⟨al⟩ takes the value \true, \false and comparisons are always done in lowercase mode (so, \true is always allowed input).

\define@boolkey[@pre]{fam}[@mp]{key}{@default}{@func}{@func2}\define@boolkey[@pre]{fam}[@mp]{key}{@default}{@func}{@func2}

This creates a boolean of the form \if[@family]{@key}³⁶ if @mp is not specified, using \newif⁷ which initiates the conditional to \iffalse and a key macro of the form \if[@family]{@key} which first checks the validity of the user input. If the input was valid, it uses it to set the boolean and afterwards, it executes @func. If the input was invalid, it will not set the boolean and xkeyval will generate an error. If @mp is specified, it will create boolean of the form \if[@mp]{@key} to compare to command keys in section 3.2. The value @default will be used by the key macro when the user didn't submit a value to the key. (See also section 3.1.)

If the + version of the macro is used, one can specify two key macros. If user input is valid, the macro will set the boolean and executes @func2. Otherwise, it will not set the boolean and execute @func.

\define@boolkey[@pre]{fam}[my@frame]{} \define@boolkey[@family]{shadows}{% 
  ifKV@family\shade 
  \PackageInfo{mypack}{turning shadows on}% 
  else 
  \PackageInfo{mypack}{turning shadows off}% 
  fi 
}\PackageWarning{mypack}{erroneous input ignored}%

⁶When you want to use this macro directly, either make sure that neither of the input parameters contain characters with a catcode different from 11 (hence no - for instance), reset the catcode of the offending characters internally to 11 or use \csname...\endcsname to construct macro names, (for instance, \csname ifpre@name-famkey\endcsname). See for more information section 8.

⁷The \toks of implementation \newif is used because it can be used in the replacement text of a macro, whereas the plain \toks\newif is defined \outer.
The first example creates the boolean \ifmy@frame and defines the key macro \ifmy@frame to only set the boolean (if input is correct). The second key informs the user about changed settings or produces a warning when input was incorrect.

One can also define multiple boolean keys with a single command.

\define@boolkeys This macro creates a boolean key for every entry in the list (keys). As with the command \define@cmdkeys, the individual keys cannot have a custom function. The boolean keys created with this command are only meant to set the state of the boolean using the user input. Concluding,

\define@boolkeys{fam}{my@}{keya,keyb,keyc}
is an abbreviation for

\define@boolkey{fam}{my@}{keya}{}
\define@boolkey{fam}{my@}{keyb}{}
\define@boolkey{fam}{my@}{keyc}{}

3.5 Checking keys

\key@undefined This macro executes (undefined) when (key) is not defined in a family listed in (families) using (prefix) (which is \K by default) and (defined) when it is. If (defined) is executed, \K@fam holds the first family in the list (families) that holds (key). If (undefined) is executed, \K@fam contains the last family of the list (families).

\key@undefined{my@}{fam1,fam2}{keys}{'keys' not defined}{'keys' defined}

This example results in ‘keya’ defined and \K@fam holds fam1y1a.

3.6 Disabling keys

It is also possible to disable keys after use as to prevent the key from being used again.

\disable@keys This when you disable a key, the use of this key will produce a warning in the log file. Disabling a key that hasn’t been defined will result in an error message.

\disable@keys{my@}{fam1a}{keys}{keya}{keyb}

This would make keya and keyb produce a warning when one tries to set these keys.

4 Setting keys

4.1 The user interface

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 \K by default.
\setkeys{[prefix]}{(families)}{(na)}{(keys)}

\setkeys
This macro sets keys of the form \( \{\text{prefix}\}\{\text{family}\}\{\text{key}\} \) where \( \text{family} \) is an element of the list \( \{\text{families}\} \) and \( \text{key} \) is an element of the options list \( \{\text{keys}\} \) and not of \( \{\text{na}\} \). 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 \( \{\text{families}\} \), the first family from the list defining the key will set it. No errors are produced when \( \{\text{keys}\} \) is empty. If \( \{\text{family}\} \) is empty, the macro will set keys of the form \( \{\text{prefix}\}\{\text{key}\} \). If \( \{\text{prefix}\} \) is specified and empty, the macro will set keys of the form \( \{\text{family}\}\{\text{key}\} \). If both \( \{\text{prefix}\} \) and \( \{\text{family}\} \) are empty, the macro will set keys of the form \( \{\text{key}\} \).

In the example above, line 1 will set \text{keya} in family familya. This effectively means that the value \text{test} will be submitted to the key macro \textbackslash{my}@\text{familya}@\text{keya}. The next line will set \text{keyb} in familya. The last one sets \text{keyb} in family familyyb. As the keys used here, directly output their value, the above code results in typesetting the word \text{test} three times.

When input is lacking for a key, \texttt{\setkeys} will check whether there is a default value for that key that can be used instead. If that is not the case, an error message will be generated. See also section 3.

\setkeys{my}{familya}{familyb}{keya=\{some-text,other-text\}}

The first line of the example above does not generate an error as this key has been defined with a default value (see section 3.1). The second line does generate an error message. See also section 9 for all possible error messages generated by \textbackslash{xkeyval}.

When you want to use commas or equality signs in the value of a key, surround the value by braces, as shown in the example below.

\setkeys{my}{familya}{keya=\{a\},keyb=\{b\},keyc=\{c\}}

It is possible to nest \texttt{\setkeys} commands in other \texttt{\setkeys} commands or in key definitions. The following, for instance,

\define@key{my}{familyb}{keyc}{#1}
\setkeys{my}{familyb}{keyc=a}{\setkeys{my}{familya}{keya=b,\setkeys{my}{familya}{keyb=c}}}

returns a and b and c.

\setkeys*
The starred version of \texttt{\setkeys} sets keys which it can locate in the given families 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 \texttt{XKV@rm} after the use of \texttt{\setkeys*}. This list will also contain keys and values coming from nested \texttt{\setkeys*} commands in the order as they have been submitted. Keys listed in \( \{\text{na}\} \) will be ignored fully and will not be appended to the \texttt{XKV@rm} list.

\setkeys*{my}{familyb}{keya-test}

Since \text{keya} is not defined in family familyyb, the value in the example above will be stored in \texttt{\XKV@rm} (so \texttt{\XKV@rm} expands to \texttt{keya-test}) 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.

\setrmkeys*[1]{familya}

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

\setrmkeys*[2]{(prefix)}{(families)}{(na)}

The macro `\setrmkeys*` acts as the `\setkeys*` 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`.

\setkeys*[3]{my}{familyb}{keya=test}
\setrmkeys*[3]{my}{familyb}
\setrmkeys*[3]{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.

The combination of `\setkeys*` and `\setrmkeys*` can be used to construct complex macros in which, for instance, a part of the keys should be set in multiple families and the rest in another family or set of families. Instead of splitting the keys or the inputs, the user can supply all inputs in a single argument and the two macros will perform the splitting and setting of keys for your macro, given that the families are well chosen.

\setkeys*[4]{(prefix)}{(families)}{(na)}{(keys)}
\setkeys*[5]{(prefix)}{(families)}{(na)}{(keys)}
\setrmkeys*[6]{(prefix)}{(families)}{(na)}{(keys)}
\setrmkeys*[7]{(prefix)}{(families)}{(na)}
\setkeys*[8]{my}{familya, familyb}{keyb=test}

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.

\setkeys*[9]{familya, familyb}{keyb=test}

The example above sets keyb in both families.

**4.2 A few details**

Several remarks should be made with respect to processing the user input. Assuming that keya up to keyd are properly defined, one could do the following.

\setkeys{family}{keys= test a, keyb={test b,c,d}, . , keyc=end}
From values consisting entirely of a \{ \} group, the outer braces will be stripped off internally.\footnote{\texttt{skeyval} actually strips off 3 levels of braces: one by using \texttt{\keyvals\p@def} and two in internal parsings. \texttt{\keyval} strips off only 2 levels: one by using \texttt{\p@def} and one in internal parsings. This difference has not yet been shown to cause problems for existing packages or new implementations. If this appears to be a problem in the future, effort will be done to solve it.} 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 around value, \texttt{skeyval} will not terminate the value when it encounters a comma in value. For instance, see the value of key in the example above. The same holds for the equality sign. Notice further that any spaces around the characters – and , (in the top level group) are removed and that empty entries will silently be ignored. This makes the example above equivalent to the example below.

```
\setkeys{family}{keys=test a, keyb=\{test b, c, d\}, keyc=end}
```

Further, when executing a key macro, the following \texttt{skeyval} internals are available.

- \texttt{\prefixx}\footnote{\texttt{\prefixx} actually strips off 3 levels of braces: one by using \texttt{\p@def} and two in internal parsings. \texttt{\keyval} strips off only 2 levels: one by using \texttt{\p@def} and one in internal parsings. This difference has not yet been shown to cause problems for existing packages or new implementations. If this appears to be a problem in the future, effort will be done to solve it.} The prefix, for instance \texttt{my}.

- \texttt{\fams}\footnote{\texttt{\fams} actually strips off 3 levels of braces: one by using \texttt{\p@def} and two in internal parsings. \texttt{\keyval} strips off only 2 levels: one by using \texttt{\p@def} and one in internal parsings. This difference has not yet been shown to cause problems for existing packages or new implementations. If this appears to be a problem in the future, effort will be done to solve it.} The list of families to search, for instance \texttt{familya, familyb}.

- \texttt{\tfam}\footnote{\texttt{\tfam} actually strips off 3 levels of braces: one by using \texttt{\p@def} and two in internal parsings. \texttt{\keyval} strips off only 2 levels: one by using \texttt{\p@def} and one in internal parsings. This difference has not yet been shown to cause problems for existing packages or new implementations. If this appears to be a problem in the future, effort will be done to solve it.} The current family, for instance \texttt{familya}.

- \texttt{\header}\footnote{\texttt{\header} actually strips off 3 levels of braces: one by using \texttt{\p@def} and two in internal parsings. \texttt{\keyval} strips off only 2 levels: one by using \texttt{\p@def} and one in internal parsings. This difference has not yet been shown to cause problems for existing packages or new implementations. If this appears to be a problem in the future, effort will be done to solve it.} The header which is a combination of the prefix and the current family, for instance \texttt{my\texttt{\familya}}.

- \texttt{\key}\footnote{\texttt{\key} actually strips off 3 levels of braces: one by using \texttt{\p@def} and two in internal parsings. \texttt{\keyval} strips off only 2 levels: one by using \texttt{\p@def} and one in internal parsings. This difference has not yet been shown to cause problems for existing packages or new implementations. If this appears to be a problem in the future, effort will be done to solve it.} The current key name, for instance \texttt{keya}.

- \texttt{\na}\footnote{\texttt{\na} actually strips off 3 levels of braces: one by using \texttt{\p@def} and two in internal parsings. \texttt{\keyval} strips off only 2 levels: one by using \texttt{\p@def} and one in internal parsings. This difference has not yet been shown to cause problems for existing packages or new implementations. If this appears to be a problem in the future, effort will be done to solve it.} The keys that should not be set, for instance \texttt{keyc, keyd}.

You can use these internals and create, for example, dynamic options systems in which user input to \texttt{\setkeys} will be used to create new keys which can be used in the very same \texttt{\setkeys} command. The \texttt{extract} package \cite{extract} provides an example for this.

### 5 Pointers

The \texttt{skeyval} package provides a pointer mechanism. Pointers can be used to copy values of keys. Hence, one can reuse the value that has been submitted to a particular key in the value of another key. This section will first describe how \texttt{skeyval} can be made to save key values. After that, it will explain how to use these saved values again. Notice already that the commands \texttt{\savevalue}, \texttt{\gsavevalue} and \texttt{\usevalue} can only be used in \texttt{\setkeys} commands.

#### 5.1 Saving values

\texttt{\savevalue}\footnote{\texttt{\savevalue} actually strips off 3 levels of braces: one by using \texttt{\p@def} and two in internal parsings. \texttt{\keyval} strips off only 2 levels: one by using \texttt{\p@def} and one in internal parsings. This difference has not yet been shown to cause problems for existing packages or new implementations. If this appears to be a problem in the future, effort will be done to solve it.} Saving a value for a particular key can be accomplished by using the \texttt{\savevalue} command with the key name as argument.
\setkeys{my}{familya}{\savevalue{keys}=test}

This example will set keys as we have seen before, but will additionally define the macro \XV0my@familya@key@value to expand to test. This macro can be used later on by xkeyval to replace pointers. In general, values of keys will be stored in macros of the form \XV0(prefix)@family@key@value. This implies that the pointer system can only be used within the same family (and prefix). We will come back to that in section 5.2.

\gsavevalue

Using the global version of this command, namely \gsavevalue, will define the value macro \XV0my@family@key@value globally. In other words, the value macro won’t survive after a \begingroup...\endgroup construct (for instance, an environment), when it has been created in this group using \savevalue and it will survive afterwards if \gsavevalue is used.

\setkeys{my}{familya}{\gsavevalue{keys}=test}

This example will globally define \XV0my@familya@keys@value to expand to test. Actually, in most applications, package authors do not want to require users to use the \savevalue form when using the pointer system internally. To avoid this, the xkeyval package also supplies the following commands.

\savekeys[(prefix)]{(family)}{(keys)}
\gsavekeys[(prefix)]{(family)}{(keys)}

The \savekeys macro stores a list of keys for which the values should always be saved to a macro of the form \XV0(prefix)@family@save. This will be used by \setkeys to check whether a value should be saved or not. The global version will define this internal macro globally so that the settings can escape groups (and environments). The \savekeys macro works incrementally. This means that new input will be added to an existing list for the family at hand if it is not in yet.

\savekeys{my}{familya}{keys, keyc}
\savekeys{my}{familya}{keyb, keyc}

The first line stores keya, keyc to \XV0my@familya@save. The next line changes the content of this macro to keya, keyc, keyb.

\delkeys[(prefix)]{(family)}{(keys)}
\gdelkeys[(prefix)]{(family)}{(keys)}
\unskeys[(prefix)]{(family)}
\gunskeys[(prefix)]{(family)}

The \delkeys macro can be used to remove some keys from an already defined list of save keys. No errors will be raised when one of the keys in the list \keys was not in the list. The global version \gdelkeys does the same as \delkeys, but will also make the resulting list global. The \unskeys macro can be used to clear the entire list of key names for which the values should be saved. The macro will make \XV0(prefix)@family@save undefined. \gunskeys is similar to \unskeys but makes the internal macro undefined globally.

\savekeys{my}{familya}{keys, keyb, keyc}
\delkeys{my}{familya}{keyb}
\unskeys{my}{familya}
The first line of this example initializes the list to contain keya, keyb, keyc. The second line removes keyb from this list and hence keya, keyc remains. The last line makes the list undefined and hence clears the settings for this family.

It is important to notice that the use of the global version \gsavekeys will only have effect on the definition of the macro \XKV\{prefix\}0\{family\}save. It will not have an effect on how the key values will actually be saved by \setkeys. To achieve that a particular key value will be saved globally (like using \gsavevalue), use the \global specifier in the \savekeys argument.

\savekeys{my}\{familya\}{keya, \global{keyc}}

This example does the following. The argument keya, \global{keyc} is saved (locally) to \XKV\{my\}0\{familya\}save. When keyc is used in a \setkeys command, the associated value will be saved globally to \XKV\{my\}0\{familya\}0\{keya\}value. When keya is used, its value will be saved locally.

All macros discussed in this section for altering the list of save keys only look at the key name. If that is the same, old content will be overwritten with new content, regardless whether \global has been used in the content. See the example below.

\savekeys{my}{familya}{\global{keyb}, keyc}
\deletekeys{my}{familya}{keyb}

The first line changes the list in \XKV\{my\}0\{familya\}0\{save from keya, \global{keyc} to keya, keyc, \global{keyb}. The second line changes the list to keya, keyc.

5.2 Using saved values

The syntax of a pointer is \usevalue{keyname} and can only be used inside \setkeys and friends. xkeyval will replace a pointer by the value that has been saved for the key that the pointer is pointing to. If no value has been saved for this key, an error will be raised. The following example will demonstrate how to use pointers (using the keys defined in section 3.1).

\setkeys{my}{familya}\{\usevalue{keya}=test\}
\setkeys{my}{familya}\{keyb=\usevalue{keya}\}

The value submitted to keyb points to keya. This has the effect that the value recorded for keya will replace \usevalue{keya} and this value (here test) will be submitted to the key macro of keyb.

Since the saving of values is prefix and family specific, pointers can only locate values that have been saved for keys with the same prefix and family as the key for which the pointer is used. Hence this

\setkeys{my}{familya}\{\usevalue{keya}=test\}
\setkeys{my}{familya}\{keyb=\usevalue{keya}\}
\setkeys{my}{familya}\{keyc=\usevalue{keyb}\}

will never work. An error will be raised in case a key value points to a key for which the value cannot be found or has not been stored.

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

\setkeys{my}{familya}\{\usevalue{keya}=test\}
\setkeys{my}{familya}\{\usevalue{keyb}=\usevalue{keya}\}
\setkeys{my}{familya}\{keyc=\usevalue{keyb}\}
This works as follows. First \keyval records the value \texttt{test} in a macro. Then, \keyb uses that value. Besides that, the value submitted to \keyb, namely \texttt{usevalue\{keya\}} will be recorded in another macro. Finally, \keyc will use the value recorded previously for \keyb, namely \texttt{usevalue\{keya\}}. That in turn points to the value saved for \keya and that value will be used.

It is important to stress that the pointer replacement will be done before \LaTeX or \TeX performs the expansion of the key macro and its argument (which is the value that has been submitted to the key). This allows pointers to be used in almost any application. (The exception is grouped material, to which we will come back later.) When programming keys (using \texttt{\define@key} and friends), you won't have to worry about the expansion of the pointers which might be submitted to your keys. The value that will be submitted to your key macro in the end, will not contain pointers. These have already been expanded and been replaced by the saved values.

A word of caution is necessary. You might get into an infinite loop if pointers are not applied with care, as the examples below show. The first example shows a direct back link.

\begin{verbatim}
\netkeys[my]{familya}{\makevalue{keys}={usevalue{keya}}}
\end{verbatim}

The second example shows an indirect back link.

\begin{verbatim}
\netkeys[my]{familya}{\makevalue{keys}={test}}
\netkeys[my]{familya}{\makevalue{keyb}={usevalue{keya}}}
\netkeys[my]{familya}{\makevalue{keys}={usevalue{keyb}}}
\end{verbatim}

In these cases, an error will be issued and further pointer replacement is canceled.

As mentioned already, pointer replacement does not work inside grouped material, \texttt{\{ ... \}}, if this group is not around the entire value (since that will be stripped off, see section 1). The following, for instance, will not work.

\begin{verbatim}
\netkeys[my]{familya}{\makevalue{keys}={test}}
\netkeys[my]{familya}{keyb=parbox{2cm}{usevalue{keys}}}
\end{verbatim}

The following provides a working alternative for this situation.

\begin{verbatim}
\netkeys[my]{familya}{\makevalue{keys}={test}}
\netkeys[my]{familya}{keyb={parbox{2cm}{usevalue{keys}}}end{minipage}}
\end{verbatim}

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

\begin{verbatim}
\netkeys[my]{familya}{\makevalue{keys}={test}}
\netkeys[my]{familya}{keyb=parbox{2cm}{\define@key fam\{familya\keys\value\}}}
\end{verbatim}

When no value has been saved for \keya, the macro \texttt{\define\familya@keya@value} is undefined. Hence one might want to do a preliminary check to be sure that the macro exists.

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

\begin{verbatim}
\define@key fam\{keys\}{keys: #1}
\define@key fam\{keyb\}{\makevalue{keys}{keyb: #1}}
\define@key fam\{keyc\}{\makevalue{keyb}{keyc: #1}}
\netkeys[fm]{\makevalue{keys}={test}}
\netkeys[fm]{\makevalue{keyb}={usevalue\{keya\}}}
\netkeys[fm]{\makevalue{keyc}={usevalue\{keyb\}}}
\netkeys[fm]{\makevalue{keys}={test}}
\netkeys[fm]{\makevalue{keyb}={usevalue\{keya\}}}
\netkeys[fm]{\makevalue{keyc}={usevalue\{keyb\}}}
\end{verbatim}
Since user input is lacking in the final two commands, the default value defined for those keys will be used. In the first case, the default value points to keysa, which results in the value tests. In the second case, the pointer points to keysb, which points to keysc (since its value has been saved now) and hence also in the final command, the value tests will be submitted to the key macro of keysd.

### 6 Presetting keys

In contrast to the default value system where users are required to specify the key without a value to assign it its default value, the presetting system does not require this. Keys which are preset will be set automatically by \setkeys when the user didn’t use those keys in the \setkeys command. When users did use the keys which are also preset, \setkeys will avoid setting them again. This section again uses the key definitions of section 3.1 in examples.

```latex
\setkeys{\prefix}{\family}{\headkeys}{\tailkeys}
\gsetkeys{\prefix}{\family}{\headkeys}{\tailkeys}
```

This macro will save \headkeys to \setkeys and \tailkeys to \setkeys. Savings are done locally by \setkeys and globally by \gsetkeys (compare \savekeys and \gsavekeys, section 5.1). The saved macros will be used by \setkeys, when they are defined, whenever \family is used in the \families argument of \setkeys. Head keys will be set before setting user keys, tail keys will be set afterwards. However, if a key appears in the user input, this particular key will not be set by any of the preset keys.

The macros work incrementally. This means that new input for a particular key replaces already present settings for this key. If no settings were present yet, the new input for this key will be appended to the end of the existing list. The replacement ignores the fact whether a \savevalue or an = has been specified in the key input. We could do the following.

```latex
\setkeys{\family}{\keys=red, \savevalue{keysb}, keysd}
\gsetkeys{\family}{\savevalue{keysa}, keysb=red, keysd}
```

After the first line of the example, the macro \setkeys will contain keysa=red, \savevalue{keysb}, keysd. After the second line of the example, the macro will contain \savevalue{keysa}, keysb=red, keysd, keysd. The \tailkeys remain empty throughout the example.

```latex
\delpresetkeys{\prefix}{\family}{\headkeys}{\tailkeys}
\gdelpresetkeys{\prefix}{\family}{\headkeys}{\tailkeys}
```

These commands can be used to (globally) delete entries from the presets by specifying the key names for which the presets should be deleted. Continuing the previous example, we could do the following.

```latex
\delpresetkeys{\family}{\keys, \keysb}
```

This redefines the list of head presets \setkeys to contain keysd, keysd. As can be seen from this example, the exact use of a key name is irrelevant for successful deletion.
These commands clear the presets for \textit{family} and works just as \texttt{\unpresetkeys}. It makes \texttt{XKV@\textit{prefix}@\textit{family}@\texttt{preseteth}} and \texttt{XKV@\textit{prefix}@\textit{family}@\texttt{preshett}} undefined. The global version will make the macros undefined globally.

Two type of problems in relation to pointers could appear when specifying head and tail keys incorrectly. This will be demonstrated with two examples. In the first example, we would like to set \texttt{keys} to blue and \texttt{keyb} to copy the value of \texttt{keya}, also when the user has changed the preset value of \texttt{keya}. Say that we implement the following.

\begin{verbatim}
\navekeys[my]{familya}{keys}
\psetkeys[my]{familya}{keys=blue, keyb=\texttt{\unvalue{keys}}}
\nsetkeys[my]{familya}{keya=red}
\end{verbatim}

This will come down to executing

\begin{verbatim}
\navekeys[my]{familya}{keys}
\nsetkeys[my]{familya}{keyb=\texttt{\unvalue{keys}}, keya=red}
\end{verbatim}

since \texttt{keya} has been specified by the user. At best, \texttt{keyb} will copy a probably wrong value of \texttt{keya}. In the case that no value for \texttt{keya} has been saved before, we get an error. We observe that the order of keys in the simplified \texttt{setkeys} command is wrong. This example shows that the \texttt{keyb=\texttt{\unvalue{keys}}} should have been in the tail keys, so that it can copy the user input to \texttt{keya}.

The following example shows what can go wrong when using presets incorrectly and when \texttt{\setkeys} contains pointers.

\begin{verbatim}
\navekeys[my]{familya}{keys}
\psetkeys[my]{familya}{keya=red}
\nsetkeys[my]{familya}{keyb=\texttt{\unvalue{keys}}}
\end{verbatim}

This will come down to executing the following.

\begin{verbatim}
\navekeys[my]{familya}{keys}
\nsetkeys[my]{familya}{keyb=\texttt{\unvalue{keys}}, keya=red}
\end{verbatim}

This results in exactly the same situation as we have seen in the previous example and hence the same conclusion holds. In this case, we conclude that the \texttt{keya=red} argument should have been specified in the head keys of the \texttt{\psetkeys} command so that \texttt{keyb} can copy the value of \texttt{keya}.

For most applications, one could use the rule of thumb that preset keys containing pointers should go in the tail keys. All other keys should go in head keys. There might, however, be applications thinkable in which one would like to implement the preset system as shown in the two examples above, for instance to easily retrieve values used in the last use of a macro or environment. However, make sure that keys in that case receive an initialization in order to avoid errors of missing values.

For completeness, the working examples are below.

\begin{verbatim}
\navekeys[my]{familya}{keys}
\psetkeys[my]{familya}{keys=blue, keyb=\texttt{\unvalue{keys}}}
\nsetkeys[my]{familya}{keya=red}
\end{verbatim}

Other examples can be found in the example files. See section 13.
7 Package option processing

The macros in this section can be used to build \LaTeX class or package options systems using \texttt{xkeyval}. These are comparable to the standard \LaTeX macros without the trailing \texttt{/CG}. See for more information about these \LaTeX macros the documentation of the source [2] or a \LaTeX manual (for instance, the \LaTeX Companion [4]). The macros in this section have been built using \texttt{\define@key} and \texttt{\setkeys} and are not available to \TeX users.

The macros below allow for specifying the \texttt{(family)} (or \texttt{(families)}) as an optional argument. This could be useful if you want to define global options which can be reused later (and set locally by the user) in a macro or environment that you define. If no \texttt{(family)} (or \texttt{(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 \texttt{(prefix)}.

However, when you use a custom prefix for other keys in your package and you want to be able to set all of the keys later with a single command, you can use the custom prefix also for the class or package options system.

Note that both \texttt{[(arg)]} and \texttt{<arg>} denote optional arguments to the macros in this section. This syntax is used to identify the different optional arguments when they appear next to each other.

\begin{minted}{latex}
\DeclareOptionX{\texttt{\textbackslash DeclareOptionX\{prefix\}\langle\texttt{family}\rangle\{\texttt{key}\}\{\texttt{default}\}\{\texttt{function}\}}}
\end{minted}

\texttt{\DeclareOptionX} declares an option (i.e., a key, which can also be used later on in the package in \texttt{\setkeys} and friends). This macro is comparable to the standard \LaTeX macro \texttt{\DeclareOption}, but with this command, the user can pass a value to the option as well. Reading that value can be done by using \texttt{\#1} in \texttt{\texttt{\textbackslash function}}. This will contain \texttt{\texttt{\textbackslash default}} when no value has been specified for the key. The value of the optional argument \texttt{\texttt{\textbackslash default}} is empty by default. This implies that when the user does not assign a value to \texttt{\texttt{\textbackslash key}} and when no default value has been defined, no error will be produced. The optional argument \texttt{\texttt{\textbackslash 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.

Assuming that the file containing the example above is called \texttt{myclass.cls}, the example is equivalent to

\begin{minted}{latex}
\texttt{\textbackslash newif\texttt{\textbackslash iflandscape}}
\texttt{\textbackslash DeclareOptionX\{\texttt{landscape}\}\{\texttt{\textbackslash landscapetrue}\}}
\texttt{\textbackslash DeclareOptionX\{\texttt{parindent}\}\{20pt\}\{\texttt{\textbackslash setlength}\\texttt{parindent}\{\texttt{\#1}\}}
\end{minted}

Notice that an empty default value has been inserted by \texttt{xkeyval} for the \texttt{landscape} option. This allows for the usual \LaTeX options use like

\begin{minted}{latex}
\texttt{\def\textbackslash define@key\{\texttt{\textbackslash myclass.cls}\}\{\texttt{\textbackslash landscape}\}\{\texttt{\textbackslash landscapetrue}\}}
\texttt{\def\textbackslash define@key\{\texttt{\textbackslash myclass.cls}\}\{\texttt{\textbackslash parindent}\}\{20pt\}\{\texttt{\textbackslash setlength}\\texttt{parindent}\{\texttt{\#1}\}}}
\end{minted}

without raising \texttt{No value specified for key ‘landscape’} errors.

These examples also show that one can also use \texttt{\define@key} (or friends, see section 3) to define class or package options. The macros presented here are supplied for the ease of package programmers wanting to convert the options section of their package to use \texttt{xkeyval}.
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.

\DeclareOptionX*\{"function\} \{\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 and is basically a copy of \setkeys. 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. This macro will not use the declaration done by \DeclareOptionX* when undeclared options appear in its argument. Instead, in this case the macro will raise an error. This mimics \LaTeX's \ExecuteOptions\' behavior.

\ExecuteOptionsX\{parindent=0pt\}

This initializes \parindent to 0pt.

\ProcessOptionsX\{\prefix\}\{\families\}\{\na\}

This macro processes the keys and values passed by the user to the class or package. The optional argument \na specifies 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 macros in the user inputs (like \thepage) as will be explained in section 12.2. When used in a class file, this macro will ignore unknown keys or options. 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 starred 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. Global options that do not have a counterpart in local families of a package or class will be skipped.
It should be noted that these implementations differ from the \LaTeX implementations of \texttt{\ProcessOptions} and \texttt{\ProcessOptions*}. The difference is in copying the global options. The \LaTeX commands always copy global options if possible. As a package author doesn't know beforehand which document class will be used and with which options, the options declared by the author might show some unwanted interactions with the global options. When the class and the package share the same option, specifying this option in the \texttt{\documentclass} command will force the package to use that option as well. With \texttt{\ProcessOptionsX \keyval} offers a package author to become fully independent of the global options and be sure to avoid conflicts with any class. Have a look at the example class, style and \texttt{.tex} file below and observe the effect of changing to \texttt{\ProcessOptions*} in the style file.\footnote{See section 3.4 for information about \texttt{\defineBoolKey}.}

\begin{verbatim}
% myclass.cls
\RequirePackage{\keyval}
\defineBoolKey{myclass.cls}{
[cls]{bool}{}\ProcessOptionsX
\LoadClass{article}
\end{verbatim}

\begin{verbatim}
% mypack.sty
\defineBoolKey{mypack.sty}{
[pkg]{bool}{}\ProcessOptionsX
\end{verbatim}

\begin{verbatim}
% test.tex
\documentclass[bool=true]{myclass}
\usepackage{mypack}
\begin{document}
parindestopt
\ifcclsbool class boolean true \else class boolean false\fi\\
\ifpkgbool package boolean true \else package boolean false\fi
\end{document}
\end{verbatim}

See section 13 for more examples.

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

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

A final remark concerns the use of expandable macros in class or package options values. Due to the construction of the \LaTeX option processing mechanism, this is not possible. However, the \keyval bundle includes a patch for the \LaTeX kernel which solves this problem. See section 14.7 for more information.

8 List of macro structures

This section provides a list of all reserved internal macro structures used for key processing. Here \texttt{pre} denotes a prefix, \texttt{fam} denotes a family and \texttt{key} denotes a key. These vary per application. The other parts in internal macro names are constant. The macros with additional \texttt{XKV} prefix are protected in the sense that all \texttt{XKV} macros
disallow the use of the \XKV prefix. Package authors using xkeyval are responsible for protecting the other types of internal macros.

\pre@fam@key
Key macro. This macro takes one argument. This macro will execute the (function) of \define@key (and friends) on the value submitted to the key macro through \setkeys.

\cmdpre@fam@key
The macro which is used by \define@cmdkey to store user input in when no custom macro prefix was specified.

\ifpre@fam@key,\pre@fam@keytrue,\pre@fam@keyfalse
The conditional created by \define@boolkey with parameters pre, fam and key if no custom macro prefix was specified. The true and false macros are used to set the conditional to \iftrue and \iffalse respectively.

\pre@fam@key@default
Default value macro. This macro expands to \pre@fam@key{default value}. This macro is defined through \define@key and friends.

\XKV@pre@fam@key@value
This macro is used to store the value that has been submitted through \setkeys to the key macro (without replacing pointers).

\XKV@pre@fam@save
Contains the names of the keys that should always be saved when they appear in a \setkeys command. This macro is defined by \savekeys.

\XKV@pre@fam@preseth
Contains the head presets. These will be submitted to \setkeys before setting user input. Defined by \presetkeys.

\XKV@pre@fam@presett
Contains the tail presets. These will be submitted to \setkeys after setting user input. Defined by \presetkeys.

An important remark should be made. Most of the macros listed above will be constructed by xkeyval internally using \csname...\endcsname. Hence almost any input to the macros defined by this package is possible. However, some internal macros might be used outside xkeyval macros as well, for instance the macros of the form \ifpre@fam@key and \cmdpre@fam@key. To be able to use these macros yourself, none of the input parameters should contain non-letter characters. If you feel that this is somehow necessary anyway, there are several strategies to make things work.

Let us consider as example the following situation (notice the hyphen – in the family name).

```latex
\define@boolkey{some-fam}{myif}
\setkeys{some-fam}{myif=false}
```

Using these keys in a \setkeys command is not a problem at all. However, if you want to use the \ifKV@some-fam@myif command itself, you can do either
9 Warnings and errors

There are several points where \texttt{xkeyval} performs a check and could produce a warning or an error. All possible warnings or and error messages are listed below with an explanation. Here \texttt{pre} denotes a prefix, \texttt{name} denotes the name of a key, \texttt{fams} denotes a family, \texttt{fams} denotes a list of families and \texttt{val} denotes some value. These vary per application. Note that messages 1 to 7 could result from erroneous key setting through \texttt{\setkeys}, \texttt{\setmkeys}, \texttt{\ExecuteOptionsX} and \texttt{\ProcessOptionsX}.

1) \texttt{value 'val' is not allowed} \hspace{1cm} \textbf{(error)}
   The value that has been submitted to a key macro is not allowed. This error can be generated by either a choice or a boolean key.

2) \texttt{name' undefined in families 'fams'} \hspace{1cm} \textbf{(error)}
   The key name is not defined in the families in \texttt{fams}. Probably you mistyped \texttt{name}.

3) \texttt{no key specified for value 'val'} \hspace{1cm} \textbf{(error)}
   \texttt{xkeyval} found a value without a key, for instance something like \texttt{-value}, when setting keys.

4) \texttt{no value recorded for key 'name'} \hspace{1cm} \textbf{(error)}
   You have used a pointer to a key for which no value has been saved previously.

5) \texttt{back linking pointers; pointer replacement canceled} \hspace{1cm} \textbf{(error)}
   You were back linking pointers. Further pointer replacements are canceled to avoid getting into an infinite loop. See section 5.2.

6) \texttt{no value specified for key 'name'} \hspace{1cm} \textbf{(error)}
   You have used the key 'name' without specifying any value for it (namely, \texttt{\setkeys{fam}{name}} and the key does not have a default value. Notice that \texttt{\setkeys{fam}{name=}} submits the empty value to the key macro and that this is considered a legal value.

7) \texttt{key 'name' has been disabled} \hspace{1cm} \textbf{(warning)}
   The key that you try to set has been disabled and cannot be used anymore.
8) `XKV' prefix is not allowed  
You were trying to use the `XKV' prefix when defining or setting keys. This error can be caused by any `xkeyval` macro having an optional prefix argument.

9) key `name' undefined  
This error message is caused by trying to disable a key that does not exist. See section 3.6.

10) no save keys defined for `pre@fam0'  
You are trying to delete or undefine save keys that have not been defined yet. See section 5.1.

11) no presets defined for `pre@fam0'  
You are trying to delete or undefine presets that have not been defined yet. See section 6.

12) xkeyval loaded before `documentclass'  
Load xkeyval after `documentclass` (or in the class that is the document class). See section 7.

10 Category codes

Some packages change the catcode of the equality sign and the comma. This is a problem for `keyval` as it then does not recognize these characters anymore and cannot parse the input. This problem can play up on the background. Consider for instance the following example and note that the `graphicx` package is using `keyval` and that Turkish babel will activate the equality sign for shorthand notation.

\documentclass{article}  
\usepackage{graphicx}  
\usepackage{turkish}{babel}  
\begin{document}  
\includegraphics[scale=.5]{some.eps}  
\end{document}

The babel package provides syntax to temporarily reset the catcode of the equality sign and switch shorthand back on after using `keyval` (in the `\includegraphics` command), namely `\shorthandoff{=}` and `\shorthandon{=}`. But having to do this every time `keyval` is invoked is quite cumbersome. Besides that, it might not always be clear to the user what the problem is and what the solution.

For these reasons, `xkeyval` performs several actions with user input before trying to parse it.\(^\text{10}\) First of all, it performs a check whether the characters `=` and `,` appear in the input with unexpected catcodes. If so, the `\@selective@sanitize` macro is used to sanitize these characters only in the top level. This means that characters inside (a) group(s), `{ }`, will not be sanitized. For instance, when using Turkish babel, it is possible to use – shorthand notation even in the value of a key, as long as this value is inside a group.

\documentclass{article}  
\usepackage{graphicx}

\(^\text{10}\) Notice that temporarily resetting catcodes before reading the input to `\setkeys` won't suffice, as it will not help solving problems when input has been read before and has been stored in a token register or a macro.
\usepackage{turkish}{babel}
\usepackage{xkeyval}
\makeatletter
\define@key{fam}{key}{#1}
\begin{document}
\includegraphics[scale=0.5]{rose.png}
\setkeys{fam}{key={some -text}}
\end{document}

In the example above, the \includegraphics command does work. Further, the first equality sign in the \setkeys command will be sanitized, but the second one will be left untouched and will be typeset as babel shorthand notation.

The commands \savekeys and \disablekeys are protected against catcode changes of the comma. The commands \setkeys and \presetkeys are protected against catcode changes of the comma and the equality sign. Note that \pdfX option macros (see section 7) are not protected as \pdfX does not protect them either.

\section{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 \V@sp@def, \V@do and \V@errx, 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 spent 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 [5, 6] 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. This specialization is available in the pst-xkeyval package, which is distributed with the xkeyval bundle and is described in section 12.3. However, since a lot of authors are involved and since it requires a change of policy, the conversion of PSTricks packages might take some time. Hence, at the moment of writing, xkeyval will conflict with pst-key and the PSTricks packages still using pst-key, which are pst-ob3d, pst-stru and pst-uml.

Have a look at the PSTricks website [5] to find out if the package that you want to use has been converted already. If not, load an already converted package (like pstricks-add) after loading the old package to make them work.
12 Additional packages

12.1 xkview

The xkeyval bundle includes a viewer utility, called xkview,\textsuperscript{11} which keeps track of the keys that are defined. This utility is intended for package programmers who want to have an overview of the keys defined in their package(s). All keys defined after loading the package will be recorded in a database. It provides the following commands to display (part of) the database.

\texttt{\textbackslash xkview\{\langle options\rangle\}}

\texttt{\textbackslash xkview}

When \langle\textit{options}\rangle is empty, the entire database will be typeset in a table created with the longtable package. The columns will, respectively, contain the key name, the prefix, the family, the type (ordinary, command, choice or boolean) and the presence of a default value for every key defined after loading xkview.

\textit{options}

There are several options to control the output of this command. This set of options can be used to set up criteria for the keys that should be displayed. If a key does not satisfy one or more of them, it won't be included in the table. For instance, the following example will display all keys with family \texttt{fama}, that do not have a default value. Notice that xkview codes 'no default value' with \texttt{[none]}.

In the following examples in this section, the same preamble will be used, but will not be displayed explicitly in the examples.

\textit{columns}

One can select the columns that should be included in the table using the \texttt{columns} option. The following example includes the columns \texttt{prefix} and \texttt{family} in the table (additional to the key name column).

\texttt{\textbackslash xkview\{columns={prefix,family}\}}

\textit{vlabels}

The remaining columns are called \texttt{type} and \texttt{default}.

If you want to refer to an option, \texttt{\textbackslash xkview} can automatically generate labels using the scheme \texttt{(prefix)-(family)-(keyname)}. Here is an example.

The package can also write (part of) the database to a file. The selection of the information happens in the same way as discussed above. When specifying a filename with the option \texttt{file}, the body of the table that is displayed, will also be written to this file. Entries will be separated by \texttt{wcolsep} which is & by default and every row

\textsuperscript{11}The xkview package is contained in the file xkview.sty.
will be concluded by `weol` which is `\` by default. The output in the file can then be used as basis for a custom table, for instance in package documentation. The following displays a table in the dvi and also writes the body to `out.tex`.

```
\zkxview{file-out}
```

`out.tex` contains

```
keys&KV&fama&ordinary&\text{default}\\
keyb&KV&fama&command&\text{[none]}\\
keyc&KV&famb&choice&\text{[none]}\\
keyd&KV&famb&boolean&\text{[none]}\\
```

The following example generates a table with entries separated by a space and no end-of-line content.

```
\zkxview{file-out,wcolsep=`space,weol`}
```

Now `out.tex` contains

```
keys KV fama ordinary default:
keyb KV fama command [none]
keyc KV famb choice [none]
keyd KV famb boolean [none]
```

**option wlabels** When post-processing the table generated in this way, one might want to refer to entries again as well. When setting `wlabels` to true, the labels with names ⟨prefix⟩-⟨family⟩-⟨keyname⟩ will be in the output file. The following

```
\zkxview{file-out,wlabels=true}
```

will result in the following content in `out.tex`

```
keys&KV&fama&\text{ordinary default}:
keyb&KV&fama&\text{command [none]}:
keyc&KV&famb&\text{choice [none]}:
keyd&KV&famb&\text{boolean [none]}
```

**option view** Finally, when you only want to generate a file and no output to the dvi, set the `view` option to false.

```
\zkxview{file-out,view=false}
```

This example only generate `out.tex` and does not put a table in the dvi.

### 12.2 `xkvltxp`

The package and class option system of \LaTeX{} contained in the kernel performs some expansions while processing options. This prevents doing for instance

```
\documentclass[title=My title,author=\textsc{My},]{myclass}
```

given that `myclass` uses `xkeyval` and defines the options `title` and `author`.

This problem can be overcome by redefining certain kernel commands. These redefinitions are contained in the `xkvltxp` package.\(^\text{12}\) If you want to allow the user of your class to be able to specify expandable macros in the package options, the user will have

\(^{12}\text{The `xkvltxp` package consists of the file `xkvltxp.sty`.}\)
to do \texttt{\textbackslash requirepackage\{xkvltxp\}} on the first line of the \LaTeX\ file. If you want to offer this functionality in a package, the user can use the package in the ordinary way with \texttt{\textbackslash usepackage\{xkvltxp\}}. This package then has to be loaded before loading the package which will use this functionality. A description of the patch can be found in the source code documentation.

The examples below summarize this information. The first example shows the case in which we want to allow for macros in the \texttt{\textbackslash documentclass} command.

```latex
\begin{verbatim}
\documentclass{article}
\ifusepackage{xkvltxp}
\else
\documentclass{article}
\end{verbatim}
```

The second example shows the case in which we want to allow for macros in a \texttt{\textbackslash usepackage} command.

```latex
\begin{verbatim}
\documentclass{article}
\usepackage{xkvltxp}
\usepackage[page=\textbackslash thepage]{mypack}
\begin{document}
\end{document}
\end{verbatim}
```

Any package or class using \texttt{\textbackslash usepackage\{xkvltxp\}} to process options can take options that contain macros in their value without expanding them prematurely. However, you can of course not use macros in options which are not of the \texttt{key=value} form since they might in the end be passed on to or copied by a package which is not using \texttt{xkvltxp} to process options, which will then produce errors. Options of the \texttt{key=value} form will be deleted from \texttt{\textbackslash classoptionslist} (see section 7) and form no threat for packages loaded later on. Finally, make sure not to pass options of the \texttt{key=value} form to packages not using \texttt{\textbackslash usepackage\{xkvltxp\}} to process options since they cannot process them. For examples see section 13.

### 12.3 \texttt{pst-xkey}

The \texttt{pst-xkey} package\footnote{The \texttt{pst-xkey} package consists of the files \texttt{pst-xkey.tex} and \texttt{pst-xkey.sty}. To load \texttt{pst-xkey} \TeX\ users do \texttt{\textbackslash input \textbackslash pst-xkey} and \texttt{\textbackslash usepackage\{pst-xkey\}} or \texttt{\textbackslash RequirePackage\{pst-xkey\}}.} implements a specialized version of the options system of \texttt{xkeyval} designed for PSTricks [5, 6]. This system gives additional freedom to PSTricks package authors since they won’t have to worry anymore about potentially redefining keys of one of the many other PSTricks packages. The command \texttt{\textbackslash pset} is redefined to set keys in multiple families. Reading the documentation of the \texttt{xkeyval} package (especially section 11) first is recommended.

Keys defined in the original distribution of PSTricks have the macro structure \texttt{\textbackslash pset@somekey} (where \texttt{pset} is literal). These can be (re)defined by

```latex
\begin{verbatim}
\define\key{\pset}{\somekey}{\function}
\end{verbatim}
```

Notice especially that these keys are located in the so-called ‘empty family’. For more information about \texttt{\define\key} and friends, see section 3.

When writing a PSTricks package, let’s say \texttt{pst-new}, you should locate keys in a family which contains the name of your package. If you only need one family, you should define keys using

```latex
\begin{verbatim}
\define\key{\pset@new}{\somekey}{\function}
\end{verbatim}
```

```latex
\end{document}
```
If you want to use multiple families in your package, you can do

\define@key{psset}{ps set-a}{somekey}{function}
\define@key{psset}{ps set-b}{anotherkey}{function}

It is important that you add all of the families that you use in your package to the list in \ps@famlist. This list of families will be used by \psset to scan for keys to set user input. You can add your families to the list using

\ps@addfams{(families)}

For instance

\ps@addfams{ps set}

or

\ps@addfams{ps set-a, ps set-b}

Only one command is needed to set PSTricks keys.

\psset{(families)}{(keys)}

This command will set \( \langle \text{keys} \rangle \) in \( \langle \text{families} \rangle \) using \setkeys+ (see section 4). When \( \langle \text{families} \rangle \) is not specified, it will set \( \langle \text{keys} \rangle \) in all families in \ps@famlist (which includes the empty family for original PSTricks keys).

\psset{somekey=red, anotherkey}
\psset{ps set-b!anotherkey=green}

13 Examples and documentation

To generate the package and example files from the source, find the source of this package, the file xkeyval.dtx, in your local \LaTeX installation or on CTAN and run it with \LaTeXX.

\latex xkeyval.dtx

This will generate the package files (xkeyval.tex, xkeyval.sty, xkvltxp.sty, keyval.tex, xkvtxhdr.tex,xkvview sty, pst-xkey.tex and pst-xkey.sty) and the example files.

The file xkvec1.tex provides an example for \LaTeX users for the macros described in sections 3, 4, 5 and 6. The file xkvec2.tex provides an example for \BibTeX users for the same macros. The files xkvec3.tex, xkvec.acls, xkvecb.cls, xkvesa.sty, xkvesb.sty and xkvesc.sty together form an example for the macros described in section 7. The set of files xkvec4.tex, xkvec.acls, xkvecb.cls, xkvesa.sty, xkvesb.sty and xkvesc.sty provides an example for sections 7 and 12.2. These files also demonstrate the possibilities of interaction between packages or classes not using xkeyval and packages or classes that do use xkeyval to set options.

To (re)generate this documentation, perform the following steps.

\latex xkeyval.dtx
\latex xkeyval.dtx
\bibtex xkeyval
\makeindex -s xglo.ist -o xkeyval.gls xkeyval.glo
\makeindex -s xind.ist -o xkeyval.ind xkeyval.idx

27
14 Implementation

14.1 xkeyval.tex

Avoid loading xkeyval.tex twice.

\%<xkvtex>
\csname XKeyValLoaded\endcsname
\let\XKeyValLoaded\endinput

Adjust some catcodes to define internal macros.

\edef\XXVcatcodes{%
  \catcode`\noexpand\@\relax
  \catcode`\noexpand\-\relax
  \catcode`\noexpand\.:\relax
  \let\noexpand\XXVcatcodes=\relax
}
\catcode`\@11\relax
\catcode`\.-12\relax
\catcode`\.:12\relax

Load some basic utilities.

\input xkutils

Initializations.

\newcount\XXV@depth
\newif\ifXXV@st
\newif\ifXXV@msg
\newif\ifXXV@pl
\newif\ifXXV@knf
\newif\ifXXV@rv
\newif\ifXXV@ipx
\newif\ifXXV@reset
\let\XXV@rm\empty

Load \LaTeX primitive if necessary and provide information.

\ifx\ProvidesFile\@undefined
  \message{2014/12/03 v2.7a key=value parser (HA)}
\else
  \ProvidesFile{xkeyval.tex}[2014/12/03 v2.7a key=value parser (HA)]
\fi

\@addtofilelist{xkeyval.tex}

Warning and error macros. We redefine the keyval error macros to use the xkeyval ones.

This avoids redefining them again when we redefine the \XXV@warn and \XXV@err macros in xkeyval.sty.

\edef\XXV@warn#1\message{xkeyval warning: #1}
\edef\XXV@err#1\message{xkeyval error: #1}
\edef\XXV@errx{\XXV@err}
\let\XV@err\XV@errx
\XV@ifstar Checks whether the following token is a * or +. Use \XV@ifnextchar to perform the action safely and ignore catcodes.
\def\XV@ifstar#1\ifnextchar#1{}
\def\XV@ifplus#1\ifnextchar#1{}

\XV@makepf \{\prefix\}\{\prefix2\}
This macro creates the prefix, like prefix0 in \prefixfamily0key. First it deletes spaces from the input and checks whether it is empty. If not empty, an -sign is added. The use of the XV prefix is forbidden to protect internal macros and special macros like saved key values.
\def\XV@makepf#1\ifx#1\empty\XV@prefix@empty\fi\fi

\XV@makehd \{\family\}
Creates the header, like prefix0family0 in \prefixfamily0key. If \family{} is empty, the header reduces to prefix0.
\def\XV@makehd#1\ifx#1\empty\XV@prefix@empty\fi\fi

\XV@rstate \{\postfix\}{\postfix2}\}
Macro to save and restore xkeyval internals to allow for nesting \setkeys commands. It executes a for loop over a set of xkeyval internals and does, for instance, \let\XV@on@i\XV@na to prepare for stepping a level deeper. If \prefix2 is empty, we step a level deeper. If \prefix1 is empty, we go a level up. The non-empty argument is always \family{\expandafter{\XV@prefix\ifx\empty\XV@prefix\empty\else\XV@prefix 0\fi}}% \fi
\fi
\expandafter{\XV@prefix\ifx\empty\XV@prefix\empty\else\XV@prefix @\fi
\fi
\expandafter{\XV@prefix\ifx\empty\XV@prefix\empty\else\XV@prefix @\fi
\fi
\expandafter{\XV@prefix\ifx\empty\XV@prefix\empty\else\XV@prefix @\fi
\fi
\expandafter{\XV@prefix\ifx\empty\XV@prefix\empty\else\XV@prefix @\fi
\fi

\XV@testopta \{\function\}
\XV@testopta Tests for the presence of an optional star or plus and executes \function afterwads.
First check for an optional prefix. Afterwards, set the prefix, set the header, remove spaces from the family and execute function.

Set prefix.

Set header.

Save family name for later use.

Test for an optional prefix. Then, set the prefix, sanitize comma’s in the list of families and remove redundant spaces from this list. Finally, check for optional key names that should not be set and execute function.

Test for an optional (prefix). Then set the (prefix), sanitzize comma’s in the list of families and remove redundant spaces from this list. Finally, check for optional key names that should not be set and execute function.

Use \XV@stoptb first to find (prefix) and the (family). Then check for optional (mp) (‘macro prefix’). Next eat the (key) name and check for an optional (default) value.

\ifXV@st gives the presence of an optional default value.
\texttt{XXV0ifcmd} \{tokens\}\{macro\}\{cmd\}\{yes\}\{no\}

This macro checks whether the \{tokens\} contains the macro specification \{macro\}. If so, the argument to this macro will be saved to \{cmd\} and \{yes\} will be executed. Otherwise, the content of \{tokens\} is saved to \{cmd\} and \{no\} is executed. This macro will, for instance, be used to distinguish key and \texttt{\global{key}} and retrieve key in the latter case.

\begin{verbatim}
93 \def\XXV0ifcmd#1#2#3{%
94 \def\XXV0ifcmd#1#2#3#4%{
95 \if\exists\isnill\\nil\else
96 \expandafter\@secondoftwo
97 \fi
98 \}\%
99 }%
100 \XXV0ifcmd\#1\#2{\\nil\}\nil\%
101 \}
\end{verbatim}

\texttt{XXV0getkeyname} \{key\}={\texttt{value}}(bin)

Utility macro to retrieve the key name from \{key\} which is of the form key=value, \texttt{savevalue(key)=value} or \texttt{gsavevalue(key)=value}. Possibly without value. \texttt{ifXXV0rkey} will record whether this particular value should be saved. \texttt{ifXXV0sg} will record whether this value should be saved globally or not. The key name will be stored in \texttt{(bin)}.

\begin{verbatim}
103 \def\XXV0getkeyname#1#2{\expandafter\XXV0g@tkeyname#1=:\\nil#2}
\end{verbatim}

\texttt{XXV0g@tkeyname} \{key\}={\texttt{value}}(bin)

Use \texttt{XXV0ifcmd} several times to check the syntax of \{value\}. Save \{key\} to \{bin\}.

\begin{verbatim}
104 \long\def\XXV0g@tkeyname#1=:\\nil#3{%
105 \XXV0ifcmd#1\{savevalue#3{\XXV0rkvtrue}\XXV0sgfalse}\%
106 \XXV0ifcmd#1\{gsavevalue#3\%
107 \}\XXV0rkvtrue\XXV0sgtrue{\XXV0rkvfalse}\XXV0sgfalse}\%
108 }%
109 }
\end{verbatim}

\texttt{XXV0getsg} \{key\}(bin)

Utility macro to check whether key or \texttt{\global{key}} has been specified in \{key\}. The key name is saved to \{bin\}.

\begin{verbatim}
110 \def\XXV0getsg#1%{
111 \expandafter\XXV0ifcmd\expandafter{}\#1{\global#2\XXV0sgtrue\XXV0sgfalse}
112 }
\end{verbatim}

\texttt{XXV0define@default} \{key\}\{default\}

Defines the default value macro for \{key\} and given \texttt{XXV0header}.

\begin{verbatim}
113 \def\XXV0define@default#1%{
114 \expandafter\def\csname XXV0header#1\endcsname\expandafter{\csname XXV0header#1\endcsname\#2}}%
115 }
\end{verbatim}

\texttt{define@key} \{(prefix)\}{\texttt{family}}

Macro to define a key in a family. Notice the use of the KV prefix as default prefix. This is done to allow setting both keyval and \texttt{xkeyval} keys with a single command. This top level command first checks for an optional \{prefix\} and the mandatory \{family\}.

\begin{verbatim}
117 \def\define@key{\XXV0testoptb\XXV0define@key}
\end{verbatim}

31
\texttt{\textless \\key} \texttt{\{default\}}

Check for an optional default value. If none present, define the key macro, else continue to eat the default value.

\texttt{\textbackslash def\texttt{\textbackslash define\textbackslash key}#1} 
\texttt{%}
\texttt{\textbackslash long\textbackslash expander\textbackslash def\texttt{\textbackslash c\textbackslash s\textbackslash n\textbackslash e\textbackslash e\textbackslash r\textbackslash h\textbackslash e\textbackslash a\textbackslash d\textbackslash er\textbackslash n\textbackslash header\textbackslash \textbackslash c\textbackslash s\textbackslash n\textbackslash e\textbackslash e\textbackslash r\textbackslash h\textbackslash e\textbackslash a\textbackslash d\textbackslash er\textbackslash \textbackslash end\textbackslash c\textbackslash s\textbackslash n\textbackslash e\textbackslash e\textbackslash r\textbackslash h\textbackslash e\textbackslash a\textbackslash d\textbackslash er\textbackslash %\textbackslash %}
\texttt{"}}

\texttt{\textbackslash define\textbackslash key} \texttt{\(\{\text{default}\}\)}

Defines the key macro and the default value macro.

\texttt{\textbackslash if\textbackslash not\textbackslash \textbackslash defines\textbackslash g\textbackslash o\textbackslash y\textbackslash \textbackslash \#1\textbackslash \textbackslash \} \texttt{\textbackslash \} \texttt{\textbackslash \}}
\texttt{\textbackslash expander\textbackslash def\texttt{\textbackslash c\textbackslash s\textbackslash n\textbackslash e\textbackslash e\textbackslash r\textbackslash h\textbackslash e\textbackslash a\textbackslash d\textbackslash er\textbackslash n\textbackslash header\textbackslash \textbackslash c\textbackslash s\textbackslash n\textbackslash e\textbackslash e\textbackslash r\textbackslash h\textbackslash e\textbackslash a\textbackslash d\textbackslash er\textbackslash \textbackslash end\textbackslash c\textbackslash s\textbackslash n\textbackslash e\textbackslash e\textbackslash r\textbackslash h\textbackslash e\textbackslash a\textbackslash d\textbackslash er\textbackslash %\textbackslash %}
\texttt{\textbackslash \}}

\texttt{\textbackslash define\textbackslash cmd\textbackslash key} \texttt{\(\{\text{prefix}\}\{\text{family}\}\{\text{mp}\}\{\text{key}\}}

Define a command key. Test for optional \texttt{\textbackslash prefix}, mandatory \texttt{\textbackslash family}, optional \texttt{\textbackslash mp} 'macro prefix' and mandatory \texttt{\textbackslash key} name.

\texttt{\textbackslash defines\textbackslash cmd\textbackslash key{\textbackslash XXV\textbackslash define\textbackslash cmd\textbackslash key{cmd}}}
\definechoicekey{(key)}
Check for optional storage bins for the input and the number of the input in the list of 
allowed inputs.
\definechoicekey1{(key)}{(allowed)}
Store the storage bin and the list of allowed inputs for later use. After that, check for an 
one default value.
\definechoicekey2{(key)}{(default)}
Define the default value macro if a default value was specified.
\definechoicekey3{(function)}
Eat correct number of arguments.
\definechoicekey4{(key macro)}{(function)}
Eat one argument and pass it on to the macro that will define the key macro.
\definechoicekey5{(key macro)}{(function1)}{(function2)}
Eat two arguments and pass these on to the macro that will define the key macro. 
\definechoicekey6{(function1)} will be executed on correct input, (function2) on incorrect input.
\definechoicekey7{(key macro)}{(function)}
Create the key macros. \choosechoice will be used to check the choice and execute 
one of its mandatory arguments.
\defineboolkey +[{(prefix)}]{(family)}{mp}{key}
Define a boolean key. This macro checks for an optional +, an optional (prefix), the
mandatory (family), an optional (mp) (macro prefix) and the mandatory (key) name.

\defineboolkey \{key\}{{default}}
Decide to eat 1 or 2 mandatory arguments for the key macro. Further, construct the
control sequence for the key macro and the one for the if.

\defineboolkey \{key\}{{default}}
Eat one mandatory key function and pass it. Insert ‘setting the if’.

\defineboolkey \{key\}{{default}}{function}
Eat two mandatory key functions and pass them. Insert ‘setting the if’.

\defineboolkey \{key\}{{default}}{function}
Create the if, the default value macro (if a default value was present) and the key macro.
We use \checkchoice internally to check the input and \res to store the
user input and pass it to setting the conditional.

\defineboolkeys \{prefix\} \{family\} \{mp\} \{keys\}
Define multiple boolean keys without user specified key function. The key will, of
course, still set the if with user input.
Loop over the list of \(<\text{keys}\rangle\) and create a boolean key for every entry.

Use \(<\text{default}\rangle\) internally to define the if, the default value macro (if present) and the key macro.

This macro is used inside key macros to perform input checks. This is the user interface to \(<\text{check_choice}\rangle\) and we only use the latter internally to avoid slow parsings of optional \(*\) and +.

Check whether \(<\text{bin}\rangle\) contains at least one control sequence and converts \(<\text{input}\rangle\) and \(<\text{allowed}\rangle\) to lowercase if requested. If \(<\text{bin}\rangle\) is empty, perform the fast \(<\text{in}\rangle\) check immediately. Else, determine whether the bin contains one or two tokens. For the first alternative, we can still use the fast \(<\text{in}\rangle\) check. Notice that this macro uses settings for \(<\text{ifXXV0int}\rangle\) and \(<\text{ifXXV0pl}\rangle\).

Check whether \(<\text{bin2}\rangle\) is empty. In that case, only the \(<\text{input}\rangle\) should be saved and we can continue with the fast \(<\text{in}\rangle\) check. If not, also the number of the input in the \(<\text{allowed}\rangle\) list should be saved and we need to do a slower while type of loop.
\XVCchcr2ckch\{\bin\}\{\input\}\{\(\text{allowed}\)\}
Checks whether \(\text{input}\) is in the list \(\text{allowed}\) and perform actions accordingly.

232 \texttt{def \XVCchcr2ckch\#1\#2\#3}\%
233 \texttt{def \XVCtmpa\#1}\%

If we have a \(\bin\), store the input there.

234 \texttt{if \XVCtmpa\#0nil\let \XVCtmpa\empty \else}
235 \texttt{def \XVCtmpa\{def\#1\#2\%}
236 \texttt{fi}
237 \texttt{in\#0\{\#2\}\{\#3\}\%}
238 \texttt{ifin\%}

The \(\text{input}\) is allowed.

239 \texttt{ifXXV0pl}

If we have a +, there are two functions. Execute the first.

240 \texttt{\XVCaddtomacro\#0nil \XVCtmpa\@firstoftwo}
241 \texttt{\else}

Else, we have one function; execute it.

242 \texttt{\XVCaddtomacro\#0nil \XVCtmpa\@firstofone}
243 \texttt{fi}
244 \texttt{\else}

If we have a +, there are two functions. Execute the second.

245 \texttt{ifXXV0pl}
246 \texttt{\XVCaddtomacro\#0nil \XVCtmpa\@secondoftwo}
247 \texttt{\else}

Else, raise an error and gobble the one function.

248 \texttt{\XVCtoks\#2\%}
249 \texttt{\XVCerr\{value \("\text{the} \XVCtoks\) is not allowed\}\%}
250 \texttt{\XVCaddtomacro\#0nil \XVCtmpa\@gobble}
251 \texttt{fi}
252 \texttt{fi}
253 \texttt{\XVCtmpa}
254 }

\XVCchcr2ckchoice\(\bin1\)\(\bin2\)\{\(\text{input}\)\}\{\(\text{allowed}\)\}
Walk over the \(\text{allowed}\) list and compare each entry with the \(\text{input}\). The input is saved in \(\bin1\), the number of the \(\text{input}\) in the \(\text{allowed}\) list (starting at zero) is saved in \(\bin2\). If the \(\text{input}\) is not allowed, \(\bin2\) will be defined to contain -1.

255 \texttt{def \XVCchcr2ckchoice\#1\#2\#3\#4}\%

Save the current value of the counter as to avoid disturbing it. We don’t use a group as that takes a lot of memory and requires some more tokens (for global definitions).

256 \texttt{def \XVCtmpa\{the \count0\} \count0\#0}

The input.

257 \texttt{def \XVCtmpb\#3}\%

Define the while loop.

258 \texttt{\XVCtmpc\#0\#1,\%
259 \texttt{\def\#1\{\#1\}\%
260 \texttt{\ifx\#1\empty

36
The \textit{input} was not in \textit{allowed}. Set the number to $-1$.

\begin{verbatim}
def#1\(\$3\)\def#2\(-1\)\count0\$XXV\$tempa
\ifXXV\$pl
  \let\$XXV\$tempd\$secondoftwo
\else
  \ifXXV\$pl
    \let\$XXV\$tempd\$gobble
  \else
    \if$#1$
  \fi
  \fi
\fi
\end{verbatim}

Execute the macro for the case that input was not allowed.

\begin{verbatim}
\XXV\toks\(\$3\)%
\XXVerr\(\text{\textbackslash the\ XXV\toks is not allowed}\)%
\let\$XXV\$tempd\$gobble
\fi
\else
  \if$#1$
\fi
\fi
\end{verbatim}

We found \textit{input} in \textit{allowed}. Save the number of the \textit{input} in the list \textit{allowed}.

\begin{verbatim}
\edef\$2\{\text{\textbackslash the\ \count0}\}\count0\$XXV\$tempa
\ifXXV\$pl
  \let\$XXV\$tempd\$XXV0ch@ckch@ice
\else
  \let\$XXV\$tempd\$XXV0ch@ckch@ice0
\fi
\fi
\else
\ advances \count0\ one
\fi
\let\$XXV\$tempd\$XXV\$tempc
\fi
\fi
\end{verbatim}

Start the while loop.

\begin{verbatim}
\XXV\$tempc\$4,\$nil,\%
\end{verbatim}

\begin{verbatim}
\XXV@ch@ckch@ice  \textit{\textbackslash nil,}
\XXV@ch@ckch@ice0 \textit{Gobble remaining \textit{text} and execute the proper key function.}
\end{verbatim}

\begin{verbatim}
\def\$XXV@ch@ckch@ice0\$\$nil,\{$@\text{firstoftwo}$
\def\$XXV@ch@ckch@ice0\$\$nil,\{$@\text{firstofone}$
\end{verbatim}

\key@ifundefined This macro allows checking if a key is defined in a family from a list of families. Check for an optional prefix.

\begin{verbatim}
\def\$key@ifundefined\{\text@opt{XXV@key@ifundefined}{XXV}}
\end{verbatim}

\begin{verbatim}
\XXV@key@ifundefined \{\textit{prefix}\} \{\textit{fams}\}
\end{verbatim}

This macro is split in two parts so that \textit{XXV@p@x} can use only the main part of the macro. First we save the prefix and the list of families.

\begin{verbatim}
\def\$XXV@key@ifundefined\{\$1\}\$2\%
\XXV\makepcf\$1\%
\XXV\check\sanitize\$2\%
\expandafter\$XXV@p@def\list\expandafter\expandafter\expandafter\}
\end{verbatim}

37
\XV@key@ifundefined {(key)}
Loop over the list of families until we find the key in a family.
\def\XV@key@ifundefined{
\XV@fntrue
\XV@sp@def\XV@tkey(#1)%
Loop over possible families.
\XV@whilelist\XV@fams\XV@tfam\if\XV@knf\fi%
Set the header.
\XV@makehd\XV@tfam
Check whether the macro for the key is defined.
\if\XV@fnf
\expandafter@firstoftwo
\else
\expandafter@secondoftwo
\fi
\}
\disable@keys {\prefix} {family}
Macro that make a key produce a warning on use.
\def\disable@keys{\XV@testoptb\XV@disable@keys}
\XV@disable@keys {(keys)}
Workhorse for \disable@keys which redefines a list of key macro to produce a warn-
ing.
\def\XV@disable@keys#1{% Wolfram Language code
\XV@check@sanitize{#1}\XV@tempa
\XV@for@#1\XV@tempa\XV@tempa%
\XV@ifundefined{\XV@header\XV@tempa}{%
\XV@err\{key \textquoteleft\XV@tempa\textquoteleft undefined\%
\}
\edef\XV@tempb{%
\noexpand\XV@warn\{key \textquoteleft\XV@tempa\textquoteleft has been disabled\%
\}
\XV@ifundefined{\XV@header\XV@tempa OpenFileDialog}{%
\edef\XV@tempc\noexpand\XV@define@key{\XV@tempa}%
\}
\edef\XV@tempc\noexpand\XV@define@key{\XV@tempa []}%
\expandafter\XV@tempc\expandafter{\XV@tempb}%
\}
\presetkeys {\prefix} {family}
\gresetkeys This provides the presetting system. The macro works incrementally: keys that have
been preset before will overwrite the old preset values, new ones will be added to the end of the preset list.

```latex
\textbackslash def\textbackslash presetkeys{\textbackslash XV0@testoptb\textbackslash XV0\textbackslash presetkeys}
\textbackslash def\textbackslash \textbackslash XV0\textbackslash presetkeys{\textbackslash XV0@testtrue\textbackslash XV0@testoptb\textbackslash XV0\textbackslash presetkeys}
```

\textbackslash XV0\textbackslash presetkeys

```latex
\{\textbackslash head\ presets\}\{\textbackslash tail\ presets\}
Execute the merging macro \textbackslash XV0\textbackslash pr\textbackslash setKeys for both head and tail presets.
```

```latex
\textbackslash def\textbackslash \textbackslash XV0\textbackslash pr\textbackslash setkeys\\textbackslash \#2\{%
\textbackslash XV0\textbackslash if\textbackslash undefined\{\textbackslash XV0\textbackslash \textbackslash XV0\textbackslash \textbackslash XV0\textbackslash header\\#2\}\{%
\textbackslash XV0\textbackslash checksanitize\{\\#1\}\textbackslash XV0\textbackslash tempa
\textbackslash if\textbackslash XV0\textbackslash st\textbackslash expandafter\textbackslash global\textbackslash fi\textbackslash expandafter\textbackslash def\textbackslash \textbackslash \textbackslash XV0\textbackslash setname\ expandname\ expandafter\{\textbackslash XV0\textbackslash tempa\%
\}%
\textbackslash expandafter\textbackslash \textbackslash XV0\textbackslash merge\textbackslash \textbackslash \textbackslash XV0\textbackslash \textbackslash XV0\textbackslash header
\textbackslash \textbackslash \textbackslash \textbackslash XV0\textbackslash getkeyname
\}%
\}
```

```latex
\textbackslash del\textbackslash presetkeys
```

```latex
\textbackslash def\textbackslash \textbackslash \textbackslash del\textbackslash presetkeys\{\textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbackslash \textbacklash```
\unpresetkeys \{\prefix\} \{\family\}

\unpresetkeys Removes presets for a particular family.

\XV0unpresetkeys Undefine the preset macros. We make them undefined since this will make them appear undefined to both versions of the macro \XV0if undefined. Making the macros \relax would work in the case that no c-T\hbox{\TeX} is available (hence using \ifx\csname), but doesn’t work when c-T\hbox{\TeX} is used (and using \ifcsname).

\savekeys \{\prefix\} \{\family\}

\savekeys Store a list of keys of a family that should always be saved. The macro works incrementally and avoids duplicate entries in the list.

\XV0savekeys \{\key list\}

\XV0savekeys Check whether something has been saved before. If not, start merging.

\delsavekeys \{\prefix\} \{\family\}

\delsavekeys Remove entries from the list of save keys.

\XV0delsavekeys \{\key list\}

\XV0delsavekeys Check whether save keys are defined and if yes, start deletion algorithm. Use the macro \XV0getsg to retrieve key names from entries.
\save\endcsname{#1}\%XV@getsg
}\%
}
}

\unsavkeys
\unsavkeys
[\prefix] [\family]
Similar to \unsavkeys, but removes the ‘save keys list’ for a particular family.

\def\unsavkeys{\XXV0setfalse\XXV0setoptb\XXV0unsavkeys}
\def\unsavkeys{\XXV0strue\XXV0setoptb\XXV0unsavkeys}

\XXV0unsavkeys \unsavkeys \unsavkeys
Workhorse for \unsavkeys.

\def\XXV0unsavkeys{%
\ifXV0undefined(XXV0\XXV0header save){%
\XXV0err{no save keys defined for ‘\XXV0header’}{%%
\csname XNV0\XXV0header save\endcsname@undefined
}
}
}

\setkeys
\setkeys[\prefix] [\families]
Set keys. The starred version does not produce errors, but appends keys that cannot
be located to the list in \XXV0rm. The plus version sets keys in all families that are
supplied.

\def\setkeys{\XXV0setopta{\XXV0setoptc\XXV0setkeys}}

\XXV0setkeys
\setkeys[\no]{\key=value list}
Workhorse for \setkeys.

\long\def\XXV0setkeys[#1]{%\XXV0resb
\let\XXV0naa@empty
\XXV0checksanitizes{\no}{\XXV0resb}
\let\XXV0naa@empty

Retrieval a list of key names from the user input.

\XXV0for@\XXV0resb\XXV0tempa{%
\expandafter[\XXV0getkeyname]\XXV0tempa\nil\%XXV0tempa
\XXV0addtolist@\XXV0naa@\XXV0tempa
}
\%

\long\def\XXV0setkeys[#1]@\empty{\fi
\long\ifnum\XXV0depth=\z@\let\XXV0rm@empty\fi

Now scan the list of families for preset keys and set user input keys.

\XXV0usepresetkeys[#1]{\preset}{%
\expandafter[\XXV0getkeys]\expandafter[\XXV0resb]{\no}%
\XXV0usepresetkeys[#1]{\preset}{%
\let\CurrentOption@empty
\)
}

\XXV0usepresetkeys
\usepresetkeys[\no]{\postfix}
Loop over the list of families and check them for preset keys. If present, set them right
away, taking into account the keys which are set by the user, available in the \XXV0naa
list.

\def\XXV0usepresetkeys#1{%
\XXV0presettrue\XXV0fam\XXV0tfam{\%

This macro starts the loop over the key=value list. Do not set keys in the list \(\langle na\rangle\).

\(\langle key=value list\rangle\){\langle na\rangle}

Split key name and value (if present). If \#3 non-empty, there was no =\(\langle value\rangle\).

\(\langle key\rangle=\langle value\rangle=\#3\rangle\)\(\langle nil\rangle\)

Check for \savevalue and \gsavevalue and remove spaces from around the key name.

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

If in the \(\langle key=value list\rangle\) list, ignore the key.

If a command with a + is used, set keys in all families on the list.
Else, scan the families on the list but stop when the key is found or when the list has run out.

\XV0while{\XV0if\XV0fam{\XV0tfam{\if\XV0knf\fi}XV0makehd{\XV0tfam}
\XV0s{\XV0sys{\XV0{X3}\XV0ksys}}\}%}
\fi
\fi
\if\XV0knf
\ifXV0inpox

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

\ifx\XV0doxs\relax
For classes, ignore unknown (possibly global) options. For packages, raise the standard \BibX error.

\if\XV0current@closextension\else
\let\CurrentOption{\XV0tkey}@unknownoptionerror
\fi

Pass the option through \DeclareOptionX*.
\else\XV0doxs\fi
\else

If not in the options section, raise an error or add the key to the list in \XV0rm when \setkeys* has been used.
\if\XV0st
\XV0addtolist@o{\XV0rm}{\CurrentOption}
\else
\XV0err{'\XV0tkey' undefined in families '\XV0fams'}%
\fi
\fi
\else

Remove global options set by the document class from \Getusedoptionlist. Global options set by other packages or classes will be removed by \ProcessOptionsX*.
\if\XV0inpox\if\XV0testclass{\XV0documentclass\xp}
\expandafter[\XV0useoption]{\XV0useoption}{\CurrentOption}X
\fi\fi
\fi
\fi
\fi
\fi

\XV0s{\XV0toksys} %
{(ind)}

This macro coordinates the work of setting a key. (ind) is an indicator for the presence of a user submitted value for the key. If empty, no value was present.
\def\XV0s{\XV0toksys}{\if\XV0funde{\XV0header\XV0tkey}{}}%
\XV0knffalse

Check global setting by \savekeys to know whether or not to save the value of the key at hand.
\XV0funde{\XV0\XV0header save}{}%
Save the value of a key.

Replace pointers by saved values.

If no value was present, use the default value macro, if one exists. Otherwise, issue an error.

Save state in case the key executes or.

Execute the key.

Restore the current state.

This macro checks whether the key in macro appears in the save list in macro (save key list). Furthermore, it checks whether or not to save the key globally. In other words, that is in the list.
\XV0replacerpointers \XV0r@placepointers \{key=value list\}
Replaces all pointers by their saved values. The result is stored in \XV0@tempa. We feed the replacement and the following tokens again to the macro to replace nested pointers. It stops when no pointers are found anymore. We keep a list of pointers replaced already for this key in \XV0@resa so we can check whether we are running in circles.

\XV0default \(token\)(tokens)
This macro checks the \XV0@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 fancyverb) 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.

\XV0default\#1@\&null{%
Retrieval the first token in the macro.
\expandafter@edef@\expandafter@\XV0@tempa
\expandafter@{\expandafter@@gobble}@\string@\#1}%
Construct the name that we expect on the basis of the keyval and xkeyval syntax of default values.
\edef@\XV0@tempa@{\XV0@header@\XV0@key}%
Sanitize \XV0@tempa to reset catcodes for comparison with \XV0@tempa.
\@oneline@sanitize@\XV0@tempb
\ifx@\XV0@tempa@\XV0@tempb
If it is safe, extract the value. We temporarily redefine the key macro to save the default value in a macro. Saving the default value itself directly to a macro when defining keys
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
\edef\csname XKV@header\endcsname{\relax}
\edef\endcsname{\relax}
\csname XKV@key \endcsname{\relax}
\edef\default{\relax}
\endgroup

Save the default value to a value macro if either the key name has been entered in a \savekeys macro or the starred form has been used.

\if\ifXKV@key\else\fi\expandafter\fi\expandafter\expandafter\global\fi\expandafter\let
\csname XKV@header \endcsname{\relax}
\edef\default{\relax}
\fi

Replace the pointers.

\expandafter\expandafter\expandafter\global\fi\expandafter\expandafter\expandafter\relax
\csname \relax{\relax}\endcsname{\relax}
\edef\default{\relax}
\iffalse
\fi

\expandafter\expandafter\expandafter\global\fi\expandafter\expandafter\expandafter\relax
\csname \relax{\relax}\endcsname{\relax}
\edef\default{\relax}
\iffalse
\fi

\setrmkeys *
\setrmkeys *
\setrmkeys *

Save internal state.

\expandafter\expandafter\expandafter\global\fi\expandafter\expandafter\expandafter\relax
\csname \relax{\relax}\endcsname{\relax}
\edef\default{\relax}
\iffalse
\fi

\expandafter\expandafter\expandafter\global\fi\expandafter\expandafter\expandafter\relax
\csname \relax{\relax}\endcsname{\relax}
\edef\default{\relax}
\iffalse
\fi

\setrmkeys *
\setrmkeys *
\setrmkeys *

Set remaining keys stored in \XKV@rm. The starred 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@testopta again to handle optional arguments.

\let\XKV@setrmkeys\XKV@testopta{\XKV@testoptc\XKV@setrmkeys}

\XKV@setrmkeys

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

\def\XKV@setrmkeys[\relax][\relax]
\def\XKV@setrmkeys[\relax][\relax]
Reset catcodes.
\XKV@catcodes
\xkvtex

14.2 xkeyval.sty

Initialize the package.
\NeedsTeXFormat{LaTeX2e}[1995/12/01]
\ProvidesPackage{xkeyval}
\[2014/12/03 v2.7a package option processing (HA)]
Initializations. Load xkeyval.tex, adjust some catcodes to define internal macros and initialize the \DeclareOptionX* working macro.
\edef\XXV@catcodes{%
\catcode\noexpand\the\catcode'\relax
\catcode\noexpand\,\the\catcode',\relax
\let\noexpand\XXV@catcodes\relax
}
\catcode'\relax
\catcode',\relax
\let\XXV@doxs\relax

Warning and error macros.
\def\XXV@warn#1{\PackageWarning{xkeyval}{#1}}
\def\XXV@err#1{\PackageError{xkeyval}{#1}@ehc}

Retrieve the document class from \documentclass. 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.
\ifx\XXV@documentclass\undefined\fi{%
\filename@parse\XXV@tempa
\if\filename@undefined\filename@extension \filename@base.\filename@ext\else\fi% \edef\XXV@documentclass{%\filename@area.\filename@base.\filename@ext\}
\fi
\if\undefined\undefined\fi

If we didn’t find the document class, raise an error, otherwise filter global options.
\if\XXV@documentclass\undefined\XXV@err{xkeyval loaded before \protect\documentclass}%
\let\XXV@documentclass\empty
\let\XXV@classoptionslist\empty
\else
\let\XXV@classoptionslist\classoptionslist
47
Code to filter key=value pairs from \classoptionslist without expanding options.

\def\XXV0@tempa{%\let\classoptionslist\empty \XXV0@forfa(\@si)\XXV0@tempa%\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expandafter=\expander}
\ExecuteOptionsX{(prefix)}{(families)}{(na)}{(key=value list)}
This macro sets keys to specified values and uses \XXV@setkeys to do the job. Insert default prefix and family name if none provided. Use \XXV@optopt to handle optional arguments and reset \ifXXV@st and \ifXXV@p1 first to avoid unexpected behavior when \setkeys** (or a friend) has been used before \ExecuteOptionsX.
\def\ExecuteOptionsX{(\ifXXV@st\false\else\false\fi)\ifXXV@optopt\false\fi\XXV@setkeys}

\ProcessOptionsX*{(prefix)}{(families)}
Processes class or package using xkeyval. The starred 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 \XXV@testoptopt to handle optional arguments.
\def\ProcessOptionsX{(\ifXXV@optopt\false\fi)\XXV@testoptopt\XXV@pox}

\XXV@pox{(na)}
Workhorse for \ProcessOptionsX and \ProcessOptionsX*.
\def\XXV@pox[#1]{%  
  \let\XXV@tempa\empty
  \set\XXV@pox: indicates that we are in \ProcessOptionsX to invoke a special routine in \XXV@setkeys.
  \XXV@poxtrue
  \set\@fileswithoptions again in case no \DeclareOptionX has been used. This will be used to identify a call to \setkeys from \ProcessOptionsX.
  \let\@fileswithoptions@badrequireerror\false
  \edef\XXV@testclass{\getname\CurrentOption}\CurrentOption%}
  \if\xkeyvalloaded by the document class, initialize \@unusedoptionlist.
  \if\XV@testclass\XV@documentclass
    \let\@unusedoptionlist\XV@classoptionslist
    \if\XV@ifundefined{\ver@xxv1tpx.sty}\false\fi{\CurrentOption}{%  
      \@onelevel@sanitize@\@ unusedoptionlist
    }%  
  \else
    \else
  \fi
  \else
    \if\xkeyvalloaded by the document class, initialize \@unusedoptionlist.
    \if\XV@testclass\XV@documentclass
      \let\@unusedoptionlist\XV@classoptionslist
      \if\XV@ifundefined{\ver@xxv1tpx.sty}\false\fi{\CurrentOption}{%  
        \@onelevel@sanitize@\@ unusedoptionlist
      }%    
    \else
      \else
    \fi
    \else
      \if\xkeyvalloaded by the document class, initialize \@unusedoptionlist.
        \if\XV@testclass\XV@documentclass
          \let\@unusedoptionlist\XV@classoptionslist
          \if\XV@ifundefined{\ver@xxv1tpx.sty}\false\fi{\CurrentOption}{%  
            \@onelevel@sanitize@\@ unusedoptionlist
          }%        
        \else
          \else
        \fi
        \else
          \if\xkeyvalloaded by the document class, initialize \@unusedoptionlist.
            \if\XV@testclass\XV@documentclass
              \let\@unusedoptionlist\XV@classoptionslist
              \if\XV@ifundefined{\ver@xxv1tpx.sty}\false\fi{\CurrentOption}{%  
                \@onelevel@sanitize@\@ unusedoptionlist
              }%            
          \else
            \else
          \fi
          \else
            \if\xkeyvalloaded by the document class, initialize \@unusedoptionlist.
              \if\XV@testclass\XV@documentclass
                \let\@unusedoptionlist\XV@classoptionslist
                \if\XV@ifundefined{\ver@xxv1tpx.sty}\false\fi{\CurrentOption}{%  
                  \@onelevel@sanitize@\@ unusedoptionlist
                }%              
            \else
              \else
            \fi
            \else
              \if\xkeyvalloaded by the document class, initialize \@unusedoptionlist.
                \if\XV@testclass\XV@documentclass
                  \let\@unusedoptionlist\XV@classoptionslist
                  \if\XV@ifundefined{\ver@xxv1tpx.sty}\false\fi{\CurrentOption}{%  
                    \@onelevel@sanitize@\@ unusedoptionlist
                  }%                
              \else
                \else
              \fi
              \else
                \if\xkeyvalloaded by the document class, initialize \@unusedoptionlist.
                  \if\XV@testclass\XV@documentclass
                    \let\@unusedoptionlist\XV@classoptionslist
                    \if\XV@ifundefined{\ver@xxv1tpx.sty}\false\fi{\CurrentOption}{%  
                      \@onelevel@sanitize@\@ unusedoptionlist
                    }%                  
                \else
                  \else
                \fi
                \else
                  \if\xkeyvalloaded by the document class, initialize \@unusedoptionlist.
                    \if\XV@testclass\XV@documentclass
                      \let\@unusedoptionlist\XV@classoptionslist
                      \if\XV@ifundefined{\ver@xxv1tpx.sty}\false\fi{\CurrentOption}{%  
                        \@onelevel@sanitize@\@ unusedoptionlist
                      }%                    
                \else
                  \else
                \fi
                \else
                  \if\xkeyvalloaded by the document class, initialize \@unusedoptionlist.
                    \if\XV@testclass\XV@documentclass
                      \let\@unusedoptionlist\XV@classoptionslist
                      \if\XV@ifundefined{\ver@xxv1tpx.sty}\false\fi{\CurrentOption}{%  
                        \@onelevel@sanitize@\@ unusedoptionlist
                      }%                    
                \else
                  \else
                \fi
              \fi
            \fi
          \fi
        \fi
      \fi
    \fi
  \fi
\fi
\fi
\fi
\fi
Add current package options to the list.
\expandafter\XV@addtolist\expandafter
\XV@setname\csname opt\@\current@endcsname

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.
\def\XV@setname{\XV@setkeys[\@1]}%  
\expandafter\XV@setname\expandafter{\XV@tempa}%  

Reset the macro created by \DeclareOptionX* to avoid processing future unknown keys using \XV@doks.
\let\XV@doks=\relax

Reset the \XV@rm macro to avoid processing remaining options with \setrmkeys.
\let\XV@rm=\empty

Reset \ifXV@inopx not in \ProcessOptionsX anymore.
\XY@inopfalse

Reset \@fileswitchoptions to allow loading of classes or packages again.
\let\@fileswitchoptions=\@processedoptions\relax

\XV@useoption \{\option\} \Remove an option from \unusedoptionlist.
\def\XV@useoption#1{%  
  \def\XV@resa{#1}%
  \ifXV@undefined{\xkv@ltexp}{%  
    \onelevel@sanitize\XV@resa
  }%
  \expandtwoargs\@removeelement{\XV@resa}%
  \@processedoptionslist\@processedoptionslist
}

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

Reset catcodes.
\XV@catcodes
\xkv@latex}

14.3 keyval.tex
Since the xkeyval macros handle input in a very different way than keyval macros, it is not wise to redefine keyval primitives (like \XV@do and \XV@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.
### 14.4 

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

```latex
\%\%\%<xkvhdr>
\%\%\%\% Taken from latex.ltx.
\%\%\%
\message{2005/02/22 v1.1 xkeyval TeX header (HA)}
\def\@nil{\nil}
\def\@empty{}
\def\newif{\if}
\count0=\escapechar \escapechar=m\%ne
\let\@iffalse=false
\def\iffalse#1\iftrue
\def\iffalse#1\iffalse
\escapechar=\count0
\def\iftrue{%\%}
\def\iffalse{%\%
\expandafter\def\csname\expandafter\@gobbletwo\string\%\%\%}
\expandafter\def\@gobbletwo\string\%2\endcsname
{\let\@ifnum=}\%}
\long\def\ifnextchar#1#2#3{%\%#3\%
\let\reservedA=#1\%
\def\reservedB=#2\%
\def\reservedC=#3\%
\futurelet\@lettoken\ifnextchar\%
\def\ifnextchar{}%\%
\ifx\@lettoken\@empty\@empty token\@empty token
\let\reservedC=\reservedA
%\%
\else

%\%
\fi
\fi
\fi
\def\:={\let\@toks=} \% this makes \@toks a space token
\def:\{\@tfnch} \expandafter\def:\{\futurelet\@lettoken\@tfnch
\let\kernel@ifnextchar\@ifnextchar
\long\def\@testopt#1#2{%\%\%
\kernel@ifnextchar[#1]{#1[#2]}}
\long\def\@firstofone#1{#1}
\long\def\@gobble #1{}
\long\def\@gobbletwo #1#2{}
\def\@expandtwoargs#1#2#3{%\%\%\%\%\%\%
\edef\reservedA{\noexpand\@firstofone{#1}{#2}}\reservedA
\edef\reservedB{\noexpand\@firstofone{#2}{#3}}\reservedB
\edef\reservedC{\noexpand\@firstofone{#3}{#4}}\reservedC
\newif\ifin\infalse
\def\in#1#2#3#4{\infalse\else\intrue\fi}%
\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\infalse\.}%
```

52
14.5 \texttt{xkvutils.tex}

Avoid loading \texttt{xkvutils.tex} twice.

\verb|\edef\xkv@afterif@undefined|\verb|{|\texttt{(csname)}}\verb|{|\texttt{(undefined)}}\verb|{|\texttt{(defined)}}

Executes \texttt{(undefined)} if the control sequence with name \texttt{(csname)} is undefined, else it executes \texttt{(defined)}. This macro uses \texttt{-\LaTeX} if possible to avoid filling \texttt{\LaTeX}'s hash when checking control sequences like key macros in the rest of the package. The use of \texttt{\xkv@afterelsefi} is necessary here to avoid \texttt{\LaTeX} picking up the second \texttt{\fi} as end of the main conditional when \texttt{(csname)} is undefined. For \texttt{\xkv@afterelsefi} this \texttt{\fi} is hidden in the group used to define \texttt{\xkv@ifundefined} in branch of the case that \texttt{(csname)} is defined. Notice the following. Both versions of the macro leave the tested control sequence undefined. However, the first version will execute \texttt{(undefined)} if the control sequence is undefined or \texttt{\relax}, whereas the second version will only execute \texttt{(undefined)} if the control sequence is undefined. This is no problem for the applications in this package.
Check whether \texttt{keyval} has been loaded and if not, load \texttt{keyval} primitives and prevent \texttt{keyval} from being loaded after \texttt{xkeyval}.

\begin{verbatim}
\def\XXV@ifundefined{\texttt{ver@keyval.sty}\{}
\input keyval
\expandafter\expandafter\expandafter\def\csname ver@keyval.sty\endcsname{1999/03/16}
\}\}
\end{verbatim}

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

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

\texttt{\@XXV@for@on}  \texttt{(list)}\texttt{\{cmd\} \{ (function) \}}
Fast for-loop. \texttt{(list)} is not expanded. Entries of \texttt{(list)} will be stored in \texttt{(cmd)} and at every iteration \texttt{(function)} is executed.
Contributed by Morten Høgholm.

\begin{verbatim}
\long\def\@XXV@for@on{\texttt{#1\#2\#3{%}
\@XXV@tempa@toks{\#1}\edef\#2{\texttt{the}\@XXV@tempa@toks}\%
\if\#2\empty\break
\else\@XXV@for@break
\else\expandafter\expandafter\expandafter\@firstoftwo
\fi
{\#2\#3\#1,\texttt{nil},%}
\end{verbatim}

54
\texttt{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0}\texttt{r} \langle cmd \rangle \langle (function)\rangle \langle (entry)\rangle ,

Looping macro.

866 \texttt{\long\textbackslash def \textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash r\textbackslash #1\#2\#3,}\% 867 \texttt{\textbackslash X\textbackslash V\textbackslash 0\textbackslash tempa\textbackslash toks\textbackslash #3}\texttt{\textbackslash def\#1{\the\textbackslash X\textbackslash V\textbackslash tempa\textbackslash toks}}\% 868 \texttt{\textbackslash if\#1\textbackslash 0\textbackslash nil} 869 \texttt{\textbackslash expandafter\textbackslash @gobbletwo} 870 \texttt{\textbackslash else} 871 \texttt{\#2\textbackslash expandafter\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash r} 872 \texttt{\textbackslash fi} 873 \texttt{\#1\{\#2\}} 874 \}

\texttt{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash break} \langle text\rangle \langle 0\textbackslash nil\rangle ,

Macro to stop the for-loop.

875 \texttt{\long\textbackslash def \textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash break \#1\textbackslash 0\textbackslash nil,}\{\textbackslash fi\}

\texttt{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash o} \langle listcmd\rangle \langle cmd\rangle \langle (function)\rangle \langle listcmd\rangle

\texttt{\long\textbackslash def \textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash o\#1{\expandafter\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash r\textbackslash \textbackslash expandafter\textbackslash #1}\}

\texttt{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash en} \{\langle list\rangle \langle cmd\rangle \{\langle function\rangle\}\}

As \texttt{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash o}, but this macro will execute \langle function\rangle also when \langle list\rangle is empty. This is done to support packages that use the 'empty family', like PSTricks.

877 \texttt{\long\textbackslash def \textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash en\#1\#2\#3{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash r\#2\{\#3\}\#1,\textbackslash 0\textbackslash nil,}}\}

\texttt{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash eo} \langle listcmd\rangle \langle cmd\rangle \langle (function)\rangle

As \texttt{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash o}, but this macro will execute \langle function\rangle also when \langle listcmd\rangle is empty.

878 \texttt{\long\textbackslash def \textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash eo\#1\#2\#3\{} 879 \texttt{\def\#2{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash r\#2\{\#3\}\textbackslash expandafter\#1,\textbackslash 0\textbackslash nil,}\%
880 \}

\texttt{\textbackslash X\textbackslash V\textbackslash w\textbackslash h\textbackslash i\textbackslash l\textbackslash i\textbackslash s\textbackslash t} \langle listcmd\rangle \langle cmd\rangle \langle (if)\rangle \langle (function)\rangle

\langle listcmd\rangle is expanded once. Execution of \langle function\rangle stops when either the list has run out of elements or \langle if\rangle is not true anymore. When using \texttt{\textbackslash iftrue} for \langle if\rangle, the execution of the macro is the same as that of \texttt{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash o}, but contains an additional check at every iteration and is hence less efficient than \texttt{\textbackslash X\textbackslash V\textbackslash f\textbackslash 0\textbackslash o} in that situation.

881 \texttt{\long\textbackslash def \textbackslash X\textbackslash V\textbackslash w\textbackslash h\textbackslash i\textbackslash l\textbackslash i\textbackslash s\textbackslash t\#1\#2\#3\textbackslash if\#4\{\%
882 \texttt{\#3\textbackslash expandafter\textbackslash X\textbackslash V\textbackslash w\textbackslash h\textbackslash i\textbackslash l\textbackslash i\textbackslash s\textbackslash t\#1,\textbackslash 0\textbackslash nil,\textbackslash 0\textbackslash nil\0\0\#3\#3\textbackslash fi\#4\}\{\textbackslash fi\%
883 \}

\texttt{\textbackslash X\textbackslash V\textbackslash w\textbackslash h\textbackslash i\textbackslash l\textbackslash i\textbackslash s\textbackslash t} \langle entry\rangle,\langle text\rangle \langle 0\rangle \langle cmd\rangle \langle (if)\rangle \langle (function)\rangle \langle (previous)\rangle

Performs iteration and checks extra condition. This macro is not optimized for the case that the list contains a single element. At the end of every iteration, the current \langle entry\rangle will be stored in \langle previous\rangle for the next iteration. The previous entry is necessary when stepping out of the loop.

884 \texttt{\long\textbackslash def \textbackslash X\textbackslash V\textbackslash w\textbackslash h\textbackslash i\textbackslash l\textbackslash i\textbackslash s\textbackslash t\#1,\#2\0\0\#3\#4\textbackslash fi\#5\#6\{\%
885 \texttt{\def\#3\{\#1\}}\%

55
If we find the end of the list, stop.

886  \[if\#3\#0nil
887  \[def\#3\#6]\[expandafter]'XXV\@oh\@list
888  \[else
889  \#4\%
890  \#5]\[expandafter]\[expandafter]\[expandafter]'XXV\@oh\@list
891  \[else
892  \[def\#3\#6]\[expandafter]\[expandafter]\[expandafter]'XXV\@oh\@list
893  \[fi
894  \[fi
895  \#2\#0#3\#4\[fi\#5\{#1\%
896  \]

\[XXV\@oh\@list\] 'text' \#0\{cmd\}\{if\}\{fi\}\{function\}\{previous\}
Macro to gobble remaining input.
897  \\long\[def]'XXV\@oh\@list\#1\#0\#2\#3'\#4\#5\}

\[XXV\@adddtomacro0n \{macro\}\{\{content\}\}
Adds \{content\} to \{macro\} without expanding it.
898  \\long\[def]'XXV\@adddtomacro0n\#1\#2\%
899  \XXV\@tempa\@toks\[expandafter]\#1\#2\%
900  \[edef\#1\{the\}XXV\@tempa\@toks\%
901 \}

\[XXV\@adddtomacro0o \{macro\}\{\{content\}\}
Adds \{content\} to \{macro\} after expanding the first token of \{content\} once. Often used to add the content of a macro to another macro.
902  \[def]'XXV\@adddtomacro0o\#1\#2\%
903  \[expandafter]'XXV\@adddtomacro0n\XXV\@adddtomacro0n\#1expandafter\#2\%
904 \}

\[XXV\@addtolist0n \{cmd\}\{\{content\}\}
Adds \{content\} to the list in \{cmd\} without expanding \{content\}. Notice that it is assumed that \{cmd\} is not undefined.
905  \[def]'XXV\@addtolist0n\#1\#2\%
906  \[if\#1\#empty
907  \XXV\@adddtomacro0n\#1\#2\%
908  \[else
909  \XXV\@adddtomacro0n\#1\#2\%
910  \[fi
911 \}

\[XXV\@addtolist0o \{cmd\}\{\{content\}\}
Adds \{content\} to the list in \{cmd\} after expanding the first token in \{content\} once.
912  \[def]'XXV\@addtolist0o\#1\#2\%
913  \[if\#1\#empty
914  \XXV\@adddtomacro0o\#1\#2\%
915  \[else
916  \XXV\@adddtomacro0o\#1\[expandafter\#2\%

56
\addtolist\x
\cmd\{\langle content\rangle\}
Adds \langle content\rangle to the list in \langle cmd\rangle after a full expansion of both \langle cmd\rangle and \langle content\rangle.

```latex
\def\XXVaddtolist\x\#1#2\{edef#1\{#1\;\empty\;else,\;#1\#2\}
```

\selectivesanitize\langle level\rangle\{\langle character string\rangle\}\{\langle cmd\rangle\}
Converts selected characters, given by \langle character string\rangle, within the first-level expansion of \langle cmd\rangle to category code 12, leaving all other tokens (including grouping braces) untouched. Thus, macros inside \langle cmd\rangle do not lose their function, as it is the case with \onelevelsanitize. The resulting token list is again saved in \langle cmd\rangle.

Example: `\def\cs{~\{fi\}~}` and `\selectivesanitize{"\}\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: \langle cmd\rangle should not contain the control sequence \bgroup; however, \cname, \endcname and \egroup are possible. The optional \langle level\rangle command controls up to which nesting level sanitizing takes place inside groups; 0 will only sanitize characters in the top level, 1 will also sanitize within the first level of braces (but not in the second), etc. The default value is 10000.

```latex
\def\selectivesanitize\{\@testopt\@selectivesanitize\0\M
\def\@selectivesanitize\{\1#1\2\3
\begingroup
\count0\1\relax\advance\count0\0\one
\XXV\tocs\expandafter\#3\%
\def\#3\{#1\}\onelevel\sanitize\#3\%
\def\#3\{#3\}\the\XXV\tocs\%
\expandafter\@selectivesanitize\expandafter\#3\%
\expandafter\@selectivesanitize\expandafter\#3\%
\expandafter\endgroup\expandafter\tocs\expandafter\the\XXV\tocs\%
\edef\#3\{\the\tocs\%
\endgroup\}
```

\selectivesanitize\langle cmd\rangle\{\langle sanitized character string\rangle\}\{\langle token list\rangle\}
Performs the main work. Here, the characters in \langle sanitized character string\rangle are already converted to catcode 12, \langle token list\rangle is the first-level expansion of the original contents of \langle cmd\rangle. The macro basically steps through the \langle token list\rangle, 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 \selectivesanitize is being called recursively if \langle token list\rangle contains grouping braces.

```latex
\def\@selectivesanitize\{\@testopt\0\@selectivesanitize\1#1#2#3\%
\def\@i\{\futurelet\@tok\@i\%
\def\@ii\%
\expandafter\@i\meaning\com@tok\relax
\ifx\com@tok\empty\@selectivesanitize
\let\com@cmd\gobble
\else
\ifx\com@tok\sptoken
```
Check whether \<content\> to be saved to macro \<cmd\> unexpanded, contains the characters \/BP or \/B8 with wrong catcodes. If so, it sanitizes them before saving \<content\> to \<cmd\>.

Similar to \XXV0checksanitize, but only checks commas.
This macro first checks whether at least one \(\textit{token}\) is in \(\langle\text{character string}\rangle\). If that is the case, it checks whether the character has catcode 12. Note that the macro will conclude that the character does not have catcode 12 when it is used inside a group \{\}, but that is not a problem, as we don’t expect \(\textit{token}\) (namely, or =) inside a group, unless this group is in a key value. But we won’t worry about those characters anyway since the relevant user key macro will have to process that. Further, it is assumed that all occurrences of \(\textit{token}\) in \(\langle\text{character string}\rangle\) have the same catcode. \(\langle\text{cmd}\rangle\) is used as a temporary macro and will contain \(\langle\text{character string}\rangle\) at the end of the macro.

\[\langle\text{cmd}\rangle\{\langle\text{token}\rangle\}\langle\text{character string}\rangle\]

Check whether there is at least one \(\textit{BP}\) present. If so, try to find it. If we can’t find it, the character(s) has (or have) the wrong catcode. In that case sanitizing is necessary. This actually occurs, because the input was read by \TeX before (and for instance stored in a macro or token register).

\[\langle\text{cmd}\rangle\text{ as }\langle\text{token list}\rangle\]

Defines \(\langle\text{cmd}\rangle\) as \(\langle\text{token list}\rangle\) after removing spaces surrounding elements of the list in \(\langle\text{token list}\rangle\). So, key1, key b becomes key1, key b. This is used to remove spaces from around elements in a list. Using \texttt{\textbackslash{}map@space} for this job, would also remove the spaces inside elements and hence changing key or family names with spaces. This method is slower, but does allow for spaces in key and family names, just as \texttt{\textbackslash{}keyval} did. We need this algorithm at several places to be able to perform \texttt{\textbackslash{}in@\{key1, key2, \ldots\}}, without having to worry about spaces in between commas and key names.

\[\langle\text{cmd}\rangle\{\langle\text{token list}\rangle\}\]

This is a merging macro. For a given new item, the old items are scanned. If an old item key name matches with a new one, the new one will replace the old one. If not, the old one will be appended (and might be overwritten in a following loop). If, at the
end of the old item loop the new item has not been used, it will be appended to the end of the list. This macro works irrespective of special syntax. The \( \text{filter} \) is used to filter the key name from the syntax, eg \( \text{global}\{\text{key}\} \). All occurrences of a particular key in the existing list will be overwritten by the new item. This macro is used to make \( \text{\textbackslash savekeys} \) and \( \text{\textbackslash preersetkeys} \) incremental. The \( \text{filter} \) is \( \text{\textbackslash XKV@getksym} \) and \( \text{\textbackslash XKV@getkeyname} \) respectively.

Start the loop over the new presets. At every iteration, one new preset will be compared with old presets.

Retrieve the key name of the new item at hand.

Store the (partially updated) old list in a temp macro and empty the original macro.

Start a loop over the old list.

Retrieve the key name of the old key at hand.

If the key names are equal, append the new item to the list and record that this key should not be added to the end of the presets list.

If the key names are not equal, then just append the current item to the list.

If, after checking the old item, no old item has been overwritten then append the new item to the end of the existing list.

If requested, save the new list globally.

\[ \text{\textbackslash XKV@delete} \{\text{\textbackslash delete}\}\{\text{\textbackslash filter}\} \]
Delete entries \( \{\text{delete}\} \) by key name from a \( \{\text{list}\} \) of presets or save keys using \( \{\text{filter}\} \). For \( \text{\textbackslash delpresetkeys} \), this is the macro \( \text{\textbackslash XKV@getksym} \) and for \( \text{\textbackslash delsavekeys} \), it is the macro \( \text{\textbackslash XKV@getksym} \).

Sanitize comma's.
Copy the current list and make the original empty.
\let.XKV@tempb#1\%
\let#1\empty

Run over the current list.
\XKV@for@0\XKV@tempb\XKV@tempb\%

Get the key name to identify the current entry.
#3\XKV@tempb\XKV@tempc

If the current key name is in the list, do not add it anymore.
\ifx@empty\XKV@addtolist@o#1\XKV@tempb\fi
%

Save globally is necessary.
\ifXKV@st\global\let#1\fi
}

Finalize.
\XKVValUtil@t\catcodes
\xkvutil\}

14.6 \xkvview.sty

This section provides a small utility for package developers. It provides several macros to generate overviews of the keys that are defined in a package or a collection of packages. It is possible to get an overview for a specific family, but also to get a complete overview of all keys that have been defined after loading this package.

\NeedsTeXFormat{LaTeX2e}[1995/12/01]
\ProvidesPackage{xkvview}[
\PackageWarning{xkvview}{Unknown option \CurrentOption}]
\ProcessOptionsX

Initializations.
\newif\ifXKV@vrukey
\newif\ifXKV@coliii
\newif\ifXKV@coliiii
\newif\ifXKV@coliv
\newwrite\XKV@out
\let.XKV@db@empty

Setup options and presets.
\define@cmdkeys[XKV\]{xkvview}[XKV\]{\%
\prefix,\family,\type,\default,\file,\columns,\colsep,\weol}[@nil]
\define@boolkeys[XKV\]{xkvview}[XKV\]{\view,\vlabels,\vlabels}[true]
\prefixkeys[XKV\]{xkvview}{\prefix,\family,\type,\default,\file,\}
\columns,\colsep=&,\weol=\view,\vlabels=false,\vlabels=false}{}
\texttt{\textbackslash XXV0\textbackslash V0tabulate} \{\texttt{key}\}\{\texttt{type}\}\{\texttt{default}\}
\texttt{\textbackslash XXV0\textbackslash V0tabulate} Adds the input information to the main database in \texttt{\textbackslash XXV0\textbackslash V0db}.

\begin{verbatim}
1065 \def\XXV0tbtabulate1#1#2#3{%
1066 \def\XXV0tempa[#3]{%  
1067 \donelevel0\sanitize\XXV0tempa  
1068 \XXV0\voldolimited#1:XXV0\db(#1=\ifx\XXV0\prefix\empty\else expandafter
1069 \XXV0\voldolimited#2=\XXV0tempa)%}
1070 }
1071 \def\XXV0tb\voldolimited#1{1}  
\end{verbatim}

\texttt{\textbackslash XXV0\textbackslash define\textbackslash key} \texttt{\textbackslash XXV0\textbackslash define\textbackslash cmd\textbackslash key} \texttt{\textbackslash XXV0\textbackslash define\textbackslash ch\textbackslash 0\textbackslash ice\textbackslash key} \texttt{\textbackslash XXV0\textbackslash define\textbackslash ch\textbackslash 0\textbackslash ic\textbackslash key} \texttt{\textbackslash XXV0\textbackslash define\textbackslash one\textbackslash 0\textbackslash col\textbackslash key}

\texttt{Redefine the internals of key defining macros to record information in the database.}

\begin{verbatim}
1072 \def\XXV0\define\textbackslash key#1{%  
1073 \ifnextchar\{\XXV0\define\textbackslash key0y{#1}{%  
1074 \XXV0\define\textbackslash default{#1}{#2}%,%  
1075 \expandafter\def\csname XXV0\textbackslash header#1\textbackslash endcsname\textbackslash #1\textbackslash %
1076 }%  
1077 }
1078 \def\XXV0\define\textbackslash cmd\textbackslash key#1%  
1079 \XXV0\textbackslash tabulate{#1}{#2}{#3}%,%  
1080 \if\XXV0\textbackslash at#1\{#1\}{#2}{#3}%,%  
1081 \else\XXV0\textbackslash tabulate{#1}{#2}{#3},%  
1082 \fi
1083 \def\XXV0\define\textbackslash ch\textbackslash 0\textbackslash ice\textbackslash key#1{%  
1084 \XXV0\tabulate{#1}{#2}{#3}%,%  
1085 \if\XXV0\textbackslash at#1\{#1\}{#2}{#3}%,%  
1086 \else\XXV0\tabulate{#1}{#2}{#3}%,%  
1087 \fi
1088 \def\XXV0\define\textbackslash ch\textbackslash 0\textbackslash ic\textbackslash key#1{%  
1089 \XXV0\textbackslash tabulate{#1}{#2}{#3}%,%  
1090 \if\XXV0\textbackslash at#1\{#1\}{#2}{#3}%,%  
1091 \else\XXV0\tabulate{#1}{#2}{#3},%  
1092 \fi
1093 \def\XXV0\define\textbackslash one\textbackslash 0\textbackslash col\textbackslash key#1{%  
1094 \XXV0\tabulate{#1}{#2}{#3}%,%  
1095 \if\XXV0\textbackslash at#1\{#1\}{#2}{#3}%,%  
1096 \else\XXV0\tabulate{#1}{#2}{#3}%,%  
1097 \fi
1098 \def\XXV0\define\textbackslash ch\textbackslash 0\textbackslash ice\textbackslash key0y{#1}{#2}{#3}%,%  
1099 \XXV0\tabulate{#1}{#2}{#3}%,%  
1100 \if\XXV0\textbackslash at#1\{#1\}{#2}{#3}%,%  
1101 \else\XXV0\tabulate{#1}{#2}{#3}%,%  
1102 \fi
1103 \if\XXV0\textbackslash at#1\{#1\}{#2}{#3}%,%  
1104 \else\XXV0\tabulate{#1}{#2}{#3}%,%  
1105 \fi
1106 \csname XXV0\textbackslash header#1\textbackslash endcsname\textbackslash #1\textbackslash %
1107 }
1108 \expandafter\def\csname XXV0\textbackslash header\textbackslash endcsname\textbackslash #1\textbackslash %
1109 \expandafter\def\csname XXV0\textbackslash header\textbackslash endcsname\textbackslash #1\textbackslash %
1110 \expandafter\def\csname XXV0\textbackslash header\textbackslash endcsname\textbackslash #1\textbackslash %
1111 \expandafter\def\csname XXV0\textbackslash header\textbackslash endcsname\textbackslash #1\textbackslash %
1112 \expandafter\def\csname XXV0\textbackslash header\textbackslash endcsname\textbackslash #1\textbackslash %
1113 \expandafter\def\csname XXV0\textbackslash header\textbackslash endcsname\textbackslash #1\textbackslash %
1114 \expandafter\def\csname XXV0\textbackslash header\textbackslash endcsname\textbackslash #1\textbackslash %
\end{verbatim}

\section*{62}
The main macro. Produces a long table and/or writes to a target file.

\def\xkview#1{%
  \setkeys{XKV}{xkview=\{#1\}}%
  \if\XKV@default@\empty@\else\@onelevel\sanitize\XKV@default\fi
  \ifno\column\information\display\all\columns\fi
  \if\XKV@columns@\empty@\else\count05\fi
  \XKV@colit\true\XKV@colit\true\XKV@colit\true\XKV@colit\true\XKV@colit\true
  \else
  \count0\one
  \xexpandtwoargs\in0{prefix,}{\XKV@columns,}%
  \if\XKV@default@\one\XKV@colit\false\fi
  \xexpandtwoargs\in0{family,}{\XKV@columns,}%
  \if\XKV@default@\one\XKV@colit\false\fi
  \xexpandtwoargs\in0{type,}{\XKV@columns,}%
  \if\XKV@default@\one\XKV@colit\false\fi
  \xexpandtwoargs\in0{default,}{\XKV@columns,}%
  \if\XKV@default@\one\XKV@colit\false\fi
  \count0\one
  \xprotect\edef\XKV@tempa{\xexpand\beginlongtable\{1\}(%
  \*\the\count0\prefix\normalfont Key/\if\XKV@colit\normalfont Prefix\fi\fi\if\XKV@colit\normalfont Family/\fi\if\XKV@colit\normalfont Type/\fi\xexpand\hline
  \xexpand\endfirsthead\xexpand\multicolumn{\the\count0\{1\}(%
  \normalfont\emph\{Continued from previous page\}))))\xexpand\hline
  \xprotect\edef\XKV@tempa{\xexpand\beginlongtable\{1\}(%
  \*\the\count0\prefix\normalfont Key/\if\XKV@colit\normalfont Prefix\fi\fi\if\XKV@colit\normalfont Family/\fi\if\XKV@colit\normalfont Type/\fi\xexpand\hline
  \xexpand\endfirsthead\xexpand\multicolumn{\the\count0\{1\}(%
  \normalfont\emph\{Continued on next page\}))))\xexpand\hline
  \xexpand\endfoot
  \xexpand\endlastfoot
  \xKV@toks\xexpandafter{\XKV@tempa}%
  \fi
}
Open the target file for writing if a file name has been specified.

Parse the entire database to find entries that match the criteria.

Finish the long table and typeset it.

Close the target file.

Parse a row in the database to get individual column entries. Select the requested columns and store the table row in the token or write it to the target file.

Check whether the current entry satisfies all criteria.

If output should go to the dvi, construct the table row and add it to the token.
When writing, construct the line and write it to file. Notice that xkeyval removes braces and spaces, so xcolsep={ } won’t make a space between column entries, but xcolsep=\space will.

1197 \verbatim
1198 %\immediate\write\XVVOout{%
1199 #1\ifXVVOcolii\XVVOxcolsep#2\fi
1200 \ifXVVOcoliii\XVVOxcolsep#3\fi
1201 \ifXVVOcol\XVVOxcolsep#4\fi
1202 \ifXVVOcol\labels\string\label{#2-#3-#1}\fi
1203 \expandafter\noexpand\XVVOxcol
1204 }%\verbatim
1205 \fi
1206 \fi
1207 \}
1208 \verbatim

14.7 \verb+xkvltxp.sty+

This section redefines some kernel macros as to avoid expansions of options at several places to allow for macros in key values in class and package options. It uses a temporary token register and some careful expansions. Notice that \@unusedoptionlist is sanitized after creation by xkeyval to avoid \@removelist causing problems with macros and braces. See for more information about the original versions of the macros below the kernel source documentation [2].

1209 %<xkvltxp.patch>
1210 %
1211 \% Based on latex.ltx.
1212 %
1213 \NeedsTeXFormat{LaTeX2e}[1995/12/01]
1214 \ProvidesPackage{xkvltxp}[2014/05/26 v1.3 LaTeX kernel patch (HA)]

Load utilities needed in this package.

1215 \verbatim

Start redefining internal \LaTeX macros.

1216 \verbatim
1217 \verbatim
1218 \verbatim
1219 \verbatim
1220 \verbatim
1221 \verbatim
1222 \verbatim
1223 \verbatim
1224 \verbatim
1225 \verbatim
1226 \verbatim
1227 \verbatim
1228 \verbatim
1229 \verbatim
1230 \verbatim

65
\begin{verbatim}
\def\@OptionNotUsed{%
  \ifx@current@closextension
    \let\reserved@a@CurrentOption
    \getline@sanitize\reserved@a
    \edef\@unusedoptionlist{
      \ifx\@unusedoptionlist\empty else\@unusedoptionlist fi
    \reserved@a}
  fi
\}
\def\@useOption{%
  \let\reserved@a@CurrentOption
  \getline@sanitize\reserved@a
  \@expandtoar@@@\@removeelement\reserved@a
  \@unusedoptionlist\@unusedoptionlist
  \@cname ds@\CurrentOption\endcsname
\}

\@files\@options
\def\@files\@options#1[#2][#3][#4]{%
  \XXV@sp\def\XXV@resb(#2)%
  \ifx#1\closextension
    \@files\@optionslist\relax
  \let\@files\@optionslist\XXV@resb
  \@reserved@a%
  \@onefile\@options#3[#2][#4][#1]
  \@documentclasshook%
  \else
  \@reserved@a%
  \@onefile\@options#3[#2][#4][#1]
  fi
\else
  \def\@reserved@b#1{%
    \ifx@nil#1relax\else
      \ifx#1\relax#1\relax\else
        \@noexpand\@onefile\@options#1
        \@noexpand\@files\@options[1][#2][#3][#4]
        \fi
      \fi
    \fi
  \}
  \def\@reserved@a{\@expandafter\reserved@b
    \@reserved@a}}
\let\@reserved@a\@files\@options
\@files\@options(s}%
\end{verbatim}

A void loading twice.

14.8 \texttt{pst-xkey.tex}

Avoid loading \texttt{pst-xkey.tex} twice.

\begin{verbatim}
\%<pstyl
\end{verbatim}

\texttt{\@files\@options}s\texttt{\@files\@options}s
\end{verbatim}

\texttt{(xktxpatch)}

\section*{14.8 \texttt{pst-xkey.tex}}

\begin{verbatim}
Avoid loading \texttt{pst-xkey.tex} twice.
\end{verbatim}

\begin{verbatim}
\%<pstyl
\end{verbatim}

\texttt{\@files\@options}s\texttt{\@files\@options}s
\end{verbatim}

\texttt{(xktxpatch)}
Load \texttt{xkeyval} when not already done by \texttt{pst-xkey.sty} and provide information.

\begin{verbatim}
\def\pst@famlist {} \def\pst@addfams {} \def\psset {} \def\@psset {} 
\end{verbatim}

\texttt{\pst@famlist} Initialize the list of families.
\texttt{\pst@addfams} Adds the family to \texttt{\pst@famlist} if it was not in yet.
\texttt{\psset} Set keys. Uses \texttt{xkeyval}'s \texttt{\setkeys}.
\texttt{\@psset} This macro defined by \texttt{pstricks.tex} is internally used as a shortcut. We have to redefine this as well to avoid problems.

Finalize.

\begin{verbatim}
\end{verbatim}

\texttt{14.9 \texttt{pst-xkey.sty}}

Initialize the package.

\begin{verbatim}
\%<*psklatex> \NeedsTeXFormat{LaTeX2e}[1995/12/01] \ProvidesPackage{pst-xkey}[2005/11/25 \textit{v}1.6 package wrapper for \texttt{pst-xkey.tex} (HA)]
\end{verbatim}
Load required package.
\if\PSTXKeyLoaded\endinput\else\input pst-xkey \fi
Ignore options.
\DeclareOptionX{%
\PackageWarning{pst-xkey}{Unknown option `{\CurrentOption}`}%
}
\ProcessOptionsX
(/pxlatex)

References

Acknowledgements
The author is grateful to Carsten Heinz, Morten Høgholm, Josselin Noirel, Heiko Oberdiek, Will Robertson, Till Tantau, Herbert Voß and Joseph Wright for help and suggestions. Thanks go to Donald Arseneau for contributing the robust \selective@sanitize macro, to Morten Høgholm for contributing a fast for-loop macro and to David Carlisle for modifications to support \par. 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.

Version history
This version history displays recent changes only.

v2.0 (2005/01/30)
General: Made \setkeys nestable .............................. 1
\XVnull@addtolist@e: Simplified ................................ 56
\XVnull@addtolist@e: Simplified ................................ 56
\XVnull@default: Repaired adding extra braces when executing default value ........ 45
\XVnull@ifundefined: Made none \LaTeX\ version not leave \relax ................... 53
\XVnull@placepointers: Simplified ................................. 45
v2.1  
General: Added 'immediate' versions of several macros  
(2005/02/08)  
1
v2.2  
General: Added viewer utility  
Improved nesting mechanism  
(2005/02/11)  
1
v2.3  
General: Added choice keys  
Increased efficiency of loops  
Updated viewer utility  
(2005/02/22)  
1
v2.4  
General: Added 'default value' column to \skview tables  
Added nesting protection for conditionals  
Changed \define@boolkey to have a key function  
Extended boolean keys  
Extended choice keys  
Inserted \pst@key in \skval source  
Removed command keys  
Updated documentation and examples  
Simplified some code  
Updated \skview  
(2005/03/31)  
1
\texttt{\XV@setkeys}: Added \texttt{\global} to make \texttt{\XV@rm} survive when \texttt{\setkeys} executed in a group.  
42
\texttt{\XV@shellist}: Avoid using grouping  
55
v2.5  
General: Added \texttt{\define@boolkeys}, \texttt{\define@cmdkey} and \texttt{\define@cmdkeys}  
Restructured documentation  
Simplified \texttt{\setkeys} internals  
Solved small bug in \texttt{\setkeys} which allowed other families to take over save key or preset key settings if the key was defined in multiple families  
Updated \skview  
(2005/05/07)  
1
\texttt{\XV@define@cmdkey}: Removed \texttt{\relax}  
34
v2.5a  
\texttt{@sanitize}: Added missing '%'  
(2005/05/31)  
57
v2.5b  
\texttt{@sanitize}: Added missing \texttt{\string}  
(2005/06/20)  
32, 62
v2.5c  
General: Making retrieving document class more robust  
(2005/07/10)  
1
v2.5d  
General: Added missing \texttt{\filename} in document class retrieval in \texttt{\skval}.  
(2005/08/12)  
1
v2.5e  
General: Updated docs  
\texttt{\pasec}: Added \texttt{\ignorespaces} as in \texttt{\ps trick\_\texttt{.tex}}  
(2005/11/25)  
67
v2.5f  
\texttt{\XV@setkeys}: Added reset of \texttt{\CurrentOption}  
(2006/11/18)  
41
\texttt{\XV@erstate}: Added \texttt{\XV@setkey} and \texttt{\XV@rm} to solve bugs  
29
v2.5g  
General: Altered policy for handling \texttt{\XV@rm} in nested \texttt{\setkeys} commands: all unknown keys will be recorded, not only the once from the outermost \texttt{\setkeys}.  
\texttt{\XV@setkeys}: Removed \texttt{\global} again for consistent approach of \texttt{\XV@rm} and to allow low groups to keep \texttt{\XV@rm local}  
(2006/12/19)  
42
\XV0setkeys: Avoid reset of \XV0rm in nested \setkeys commands ............ 41
\XV0resetate: Removed \XV0rm again on user request .................... 29

v2.6  
(2008/08/10)
\protect\selective@sanitize: protecting assignments from # .... 57
General: Added \def to keyval.tex as it is used by some packages .... 50
\XV0addtomacro@: protecting assignments from # .................... 56
\XV0addtomacro@: protecting assignments from # .................... 56
\protect\selective@sanitize: protecting assignments from # ........ 59
\protect\for@: protecting assignments from # .......................... 54
\XV0for@: protecting assignments from # .............................. 54
\XV0for@: Solved bug occurring with empty prefix ..................... 62

v2.6a  
(2008/08/13)
\protect\selective@sanitize: Use private scratch register .......... 57
\XV0addtomacro@: Use private scratch register ...................... 56
\XV0addtomacro@: Use private scratch register ...................... 56
\protect\selective@sanitize: Use private scratch register ......... 59
\protect\for@: Use private scratch register .......................... 54
\XV0for@: Use private scratch register .............................. 54

v2.6b  
(2012/10/14)
\XV0resetate: Added CurrentOption to fix class options not being removed from \dunseoptionlist when \setkeys is nested in \ProcessOptionX .......... 29

v2.6c  
(2014/04/27)
General: Added support for \par to support similar changes in keyval .......... 1

v2.6d  
(2014/05/09)
General: Implemented fix conform fix in keyval ...................... 1

v2.7  
(2014/12/03)
\XV0@placepointers: Removed erroneous \empty ...................... 45

v2.7  
(2014/05/25)
General: Moved several utility macros to xkvutils .................... 1

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
\@fileswithoptions ........ 686, 1276
\@badrequireerror ........ 642, 656
\@classoptionslist ........ 618, 620, 623, 626, 1254, 1255
\@classextension ........ 464, 604, 1236, 1253
\@current ........ 464, 635, 648, 657, 680, 1236
\@curname ........ 635, 648, 657, 680
\@documentclasshook .... 1258
\@filelist ................ 602
\@fileswithoptions ........ 642, 656, 686, 1251, 1276
\@firsttoone ........ 242, 287, 791
\@firstoftwo ........ 36, 37, 98, 240, 286, 304, 818, 826, 836, 851
\@ifcharacter ........ 844
\@ifnextcharacter ........ 36, 37, 844
\@onefilewithoptions .... 1257, 1261, 1267
\@onelevel@sanitize .... 553, 661, 692, 803, 925, 988, 1067, 1128, 1238, 1246
\@passoptions ........ 1216
\@pkgextension .......... 1268
\@psset ............. 1306
\@removeelement .... 694, 1247
\@selective@sanitize .... 927, 932
\@selective@sanitize .... 920
\@seconsoftwo .... 96, 246, 263, 306, 818, 828, 834, 853
\@selective@sanitize .... 920, 980, 984
\@unknownoptionerror .... 465
\@unprocessedoptions .... 687
\@unseoptionlist ........ 659, 661, 695, 1239, 1240, 1248
\@useoption ........ 1244

70
\ps@addfam \ps@famlist \PSXKeyLoaded 27,1296
\savekeys \savevalue \setkeys 27,1295,1298,1299,1303,1307
\PTXKeyLoaded 1280,1316
\S
\savekeys 12,370
\savevalue 11,105
\setkeys 9,402,1127,1305,1307
\setkeys+ 9
\setkeys++ 10
\setkeys+ 10
\setkeys++ 10
\sremkeys 10,500
\sremkeys+ 10
\T
type (option) 24
\U
\unpresetkeys 16,358
\unsavkeys 12,392
\usevalue 13,528,530
\V
\view (option) 25
\vlabels (option) 24
\W
\wcolsep (option) 24
\wcol (option) 24
\vlabels (option) 25
\write 1197
\X
\XKeyvalLoaded 3,591,1288
\XKeyvalutilscatcodes 811,813,1041
\XKeyvalutilLoaded 810
\XVald2x 646
\XValchkckckchkicr 275,286
\XValchkckckbic 273,286
\XValchkckckchoice 229,225
\XValckx 646
\XValcmd 93
\XValc@opt 628
\XValaddtolist@n 671,905
\XValaddtolist@ 470,623,679,912,1021,1023,1026,1037
\XValaddtolist@x 408,542,919,1068
\XValaddtomacro@n 240,242,246,250,531,898,903,907,909
\XValaddtomacro@ 902,914,916,1003
\XValafterelsefi 154,174,226,495,539,570,820,822,1101
\XValafterfi 196,176,228,502,533,541,574,820,1103
\XValaddtolist@ 6,213
\XValaddtolist@ 6
\XValaddtolist@ 6
\XValaddtolist@ 6
\XValaddtolist@ 218,227,232
\XValaddtolist@ 220,224
\XValaddtolist@ 978,979,983,986
\XValaddtolist@ 167,194,198,213,214,1118,1122
\XValaddtolist@ 336,404,977,1011
\XValaddtolist@ 78,291,311,374,982,1031
\XValaddtolist@ 3,616,618,659,676
\XValaddtolist@ 161,162,163
\XValaddtolist@ 182,186,189,210,1072
\XValaddtolist@ 147,149,1072
\XValaddtolist@ 143,144
\XValaddtolist@ 119,123,1072
\XValaddtolist@ 19,123
\XValaddtolist@ 347,348,350
\XValaddtolist@ 643,646
\XValaddtolist@ 499,549
\XValaddtolist@ 172,173
\XValaddtolist@ 202,203
\XValaddtolist@ 307,134,135
\XValaddtolist@ 113,124
\XValaddtolist@ 129,150,191,1080,1086,1096,1112
\XValaddtolist@ 117,118,320,322,1072
\XValaddtolist@ 354,388,1030
\XValaddtolist@ 344,345,346
\XValaddtolist@ 382,383,384
\XValaddtolist@ 16,55
\XValaddtolist@ 61,410,503,506,571,573,575,577
\XValaddtolist@ 309,310
\XValaddtolist@ 307,314,352,362,386,396,441
\XValaddtolist@ 472,497,535,540,600,614,730,744
\XValaddtolist@ 862,866,877,879