Abstract

exPkv|DEF provides a small \( \langle\text{key}\rangle = \langle\text{value}\rangle \) interface to define keys for exPkv. Key-types are declared using prefixes, similar to static typed languages. The stylised name is expKvDef but the files use expkv-def, this is due to CTAN-rules which don’t allow | in package names since that is the pipe symbol in *nix shells.

Contents

1 Documentation 2
  1.1 Macros ................................................. 2
  1.2 Prefixes .............................................. 2
    1.2.1 p-Prefixes ....................................... 2
    1.2.2 t-Prefixes ...................................... 3
  1.3 Bugs ................................................ 6
  1.4 Example ............................................ 6
  1.5 License ............................................. 7

2 Implementation 8
  2.1 The \LaTeX Package ................................ 8
  2.2 The Generic Code .................................. 8
    2.2.1 Key Types ...................................... 10
    2.2.2 Key Type Helpers ................................. 18
    2.2.3 Tests ........................................... 18
    2.2.4 Messages ....................................... 20

Index 22

*jspratte@yahoo.de
1 Documentation

Since the trend for the last couple of years goes to defining keys for a \langle key\rangle=\langle value\rangle interface using a \langle key\rangle=\langle value\rangle interface, I thought that maybe providing such an interface for \texttt{expkv} will make it more attractive for actual use, besides its unique selling points of being fully expandable, and fast and reliable. But at the same time I don't want to widen \texttt{expkv}'s initial scope. So here it is \texttt{expkvdef}, go define \langle key\rangle=\langle value\rangle interfaces with \langle key\rangle=\langle value\rangle interfaces.

Unlike many of the other established \langle key\rangle=\langle value\rangle interfaces to define keys, \texttt{expkvdef} works using prefixes instead of suffixes (e.g., .tl_set:N of \texttt{l3keys}) or directory like handlers (e.g., .store in of \texttt{pgfkeys}). This was decided as a personal preference, more over in TeX parsing for the first space is way easier than parsing for the last one. \texttt{expkvdef}'s prefixes are sorted into two categories: \texttt{p}-type, which are equivalent to TeX's prefixes like \texttt{\long}, and \texttt{t}-type defining the type of the key. For a description of the available \texttt{p}-prefixes take a look at \texttt{subsubsection 1.2.1}, the \texttt{t}-prefixes are described in \texttt{subsubsection 1.2.2}.

\texttt{expkvdef} is usable as generic code and as a \LaTeX package. It'll automatically load \texttt{expkv} in the same mode as well. To use it, just use one of

\begin{verbatim}
\usepackage{expkv-def} \% LaTeX
\input expkv-def \% plainTeX
\end{verbatim}

1.1 Macros

Apart from version and date containers there is only a single user-facing macro, and that should be used to define keys.

\begin{verbatim}
\ekvdefinekeys
\end{verbatim}

\texttt{\ekvdefinekeys{\langle set\rangle}{\langle key\rangle=\langle value\rangle, ...}}

In \langle set\rangle, define \langle key\rangle to have definition \langle value\rangle. The general syntax for \langle key\rangle should be

\begin{verbatim}
\langle prefix\rangle \langle name\rangle
\end{verbatim}

Where \langle prefix\rangle is a space separated list of optional \texttt{p}-type prefixes followed by one \texttt{t}-type prefix. The syntax of \langle value\rangle is dependent on the used \texttt{t}-prefix.

\begin{verbatim}
\ekvDate \ekvVersion
\end{verbatim}

These two macros store the version and date of the package.

1.2 Prefixes

As already said there are \texttt{p}-prefixes and \texttt{t}-prefixes. Not every \texttt{p}-prefix is allowed for all \texttt{t}-prefixes.

1.2.1 \texttt{p}-Prefixes

The \texttt{p}-type prefixes are pretty simple by nature, so their description is pretty simple. They affect the \langle key\rangle at use-time, so omitting \texttt{long} doesn't mean that a \langle definition\rangle can't contain a \texttt{\par} token, only that the \langle key\rangle will not accept a \texttt{\par} in \langle value\rangle.
The following key will be defined \protected. Note that key-types which can't be defined expandable will always use \protected.

The following key will be defined \long.

1.2.2 \textit{t-Prefixes}

Since the \textit{p}-type prefixes apply to some of the \textit{t}-prefixes automatically but sometimes one might be disallowed we need some way to highlight this behaviour. In the following an enforced prefix will be printed black (\protected), allowed prefixes will be grey (\protected), and disallowed prefixes will be red (\protected). This will be put flush-right in the syntax showing line.

\begin{verbatim}
code \texttt{code \{key\} = \{\texttt{definition}\}} \texttt{protected \long}
\end{verbatim}

Define \texttt{\{key\}} to expand to \texttt{\{definition\}}. The \texttt{\{key\}} will require a \texttt{\{value\}} for which you can use \verb|#1| inside \texttt{\{definition\}}. The \texttt{ecode} variant will fully expand \texttt{\{definition\}} inside an \texttt{edef}.

\begin{verbatim}
noval \texttt{noval \{key\} = \{\texttt{definition}\}} \texttt{protected \long}
\end{verbatim}

The \texttt{noval} type defines \texttt{\{key\}} to expand to \texttt{\{definition\}}. The \texttt{\{key\}} will not take a \texttt{\{value\}}. \texttt{enoval} fully expands \texttt{\{definition\}} inside an \texttt{edef}.

\begin{verbatim}
default \texttt{default \{key\} = \{\texttt{definition}\}} \texttt{protected \long}
\end{verbatim}

This serves to place a default \texttt{\{value\}} for a \texttt{\{key\}} that takes an argument, the \texttt{\{key\}} can be of any argument-grabbing kind, and when used without a \texttt{\{value\}} it will be passed \texttt{\{definition\}} instead. The \texttt{qdefault} variant will expand the \texttt{\{key\}}'s code once, so will be slightly quicker, but not change if you redefine \texttt{\{key\}}. The \texttt{edefault} on the other hand fully expands the \texttt{\{key\}}-code with \texttt{\{definition\}} as its argument inside of an \texttt{edef}.

\begin{verbatim}
initial \texttt{initial \{key\} = \{\texttt{value}\}} \texttt{protected \long}
\end{verbatim}

With \texttt{initial} you can set an initial \texttt{\{value\}} for an already defined argument taking \texttt{\{key\}}. It'll just call the key-macro of \texttt{\{key\}} and pass it \texttt{\{value\}}. The \texttt{einitial} variant will expand \texttt{\{value\}} using an \texttt{edef} expansion prior to passing it to the key-macro and the \texttt{oinital} variant will expand the first token in \texttt{\{value\}} once.
bool \langle key \rangle = \langle cs \rangle

The \langle cs \rangle should be a single control sequence, such as \texttt{\iffoo}. This will define \langle key \rangle to be a boolean key, which only takes the values \texttt{true} or \texttt{false} and will throw an error for other values. If the key is used without a \langle value \rangle it’ll have the same effect as if you use \langle key \rangle = \texttt{true}. bool and gbool will behave like \TeX-if\-s so either be \texttt{\iftrue} or \texttt{\iffalse}. The bool1TF and gbool1TF variants will both take two arguments and if true the first will be used else the second, so they are always either \texttt{\@firstoftwo} or \texttt{\@secondoftwo}. The variants with a leading g will set the control sequence globally, the others locally. If \langle cs \rangle is not yet defined it’ll be initialised as the \texttt{false} version. Note that the initialisation is \texttt{not} done with \texttt{\newif}, so you will not be able to do \texttt{\@true} outside of the \langle key \rangle = \langle value \rangle interface, but you could use \texttt{\newif} yourself. Even if the \langle key \rangle will not be \texttt{protected} the commands which execute the \texttt{true} or \texttt{false} choice will be, so the usage should be safe in an expansion context (\texttt{e.g.}, you can use \texttt{edefault} \langle key \rangle = \texttt{false} without an issue to change the default behaviour to execute the \texttt{false} choice).

store \langle key \rangle = \langle cs \rangle

The \langle cs \rangle should be a single control sequence, such as \texttt{\foo}. This will define \langle key \rangle to store \langle value \rangle inside of the control sequence. If \langle cs \rangle isn’t yet defined it will be initialised as empty. The variants behave similarly to their \texttt{\edef}, \texttt{\edef}, \texttt{\gdef}, and \texttt{\xdef} counterparts, but \texttt{store} and \texttt{gstore} will allow you to store macro parameters inside of them by using \texttt{\unexpanded}.

data \langle key \rangle = \langle cs \rangle

The \langle cs \rangle should be a single control sequence, such as \texttt{\foo}. This will define \langle key \rangle to store \langle value \rangle inside of the control sequence. But unlike the \texttt{store} type, the macro \langle cs \rangle will be a switch at the same time, it’ll take two arguments and if \langle key \rangle was used expands to the first argument followed by \langle value \rangle in braces, if \langle key \rangle was not used \langle cs \rangle will expand to the second argument (so behave like \texttt{\@secondoftwo}). The idea is that with this type you can define a key which should be typeset formatted. The \texttt{edata} and \texttt{xdata} variants will fully expand \langle value \rangle, the \texttt{gdata} and \texttt{xdata} variants will store \langle value \rangle inside of \langle cs \rangle globally. The \texttt{p} prefixes will only affect the key-macro, \langle cs \rangle will always be expandable and \texttt{\long}.

dataT \langle key \rangle = \langle cs \rangle

Just like \texttt{data}, but instead of \langle cs \rangle grabbing two arguments it’ll only grab one, so by default it’ll behave like \texttt{\@gobble}, and if a \langle value \rangle was given to \langle key \rangle the \langle cs \rangle will behave like \texttt{\@firstoftwo} appended by \texttt{\{value\}}.

int \langle key \rangle = \langle cs \rangle

The \langle cs \rangle should be a single control sequence, such as \texttt{\foo}. An \texttt{int} key will be a \TeX-count register. If \langle cs \rangle isn’t defined yet, \texttt{\newcount} will be used to initialise it. The \texttt{eint} and \texttt{xint} versions will use \texttt{\numexpr} to allow basic computations in their \langle value \rangle. The \texttt{gint} and \texttt{xint} variants set the register globally.

dimen \langle key \rangle = \langle cs \rangle

The \langle cs \rangle should be a single control sequence, such as \texttt{\foo}. This is just like \texttt{int} but uses a dimen register, \texttt{\newdimen} and \texttt{\dimexpr} instead.
skip \langle key \rangle = \langle cs \rangle 

The \langle cs \rangle should be a single control sequence, such as \textbackslash foo. This is just like \texttt{int} but uses a skip register, \texttt{\newskip} and \texttt{\glueexpr} instead.

toks \langle key \rangle = \langle cs \rangle 

gtoks \langle key \rangle = \langle cs \rangle 

The \langle cs \rangle should be a single control sequence, such as \textbackslash foo. Store \langle value \rangle inside of a toks-register. The g variants use \texttt{\global}, the app variants append \langle value \rangle to the contents of that register. If \langle cs \rangle is not yet defined it will be initialised with \texttt{\newtoks}.

box \langle key \rangle = \langle cs \rangle 

gbox \langle key \rangle = \langle cs \rangle 

The \langle cs \rangle should be a single control sequence, such as \textbackslash foo. Typesets \langle value \rangle into a \texttt{\hbox} and stores the result in a box register. The boxes are colour safe. \texttt{expX} doesn’t provide a \texttt{\vbox} type.

meta \langle key \rangle = \{(key)=(value), ...\} 

This key type can set other keys, you can access the \langle value \rangle which was passed to \langle key \rangle inside the \langle key \rangle=(value) list with \#1. It works by calling a sub-\texttt{\ekvset} on the \langle key \rangle=(value) list, so a \texttt{set} key will only affect that \langle key \rangle=(value) list and not the current \texttt{\ekvset}. Since it runs in a separate \texttt{\ekvset} you can’t use \texttt{\ekvsneak} using keys or similar macros in the way you normally could.

nmeta \langle key \rangle = \{(key)=(value), ...\} 

This key type can set other keys, the difference to meta is, that this key doesn’t take a value, so the \langle key \rangle=(value) list is static.

smeta \langle key \rangle = \{(set)\{(key)=(value), ...\} 

Yet another meta variant. An smeta key will take a \langle value \rangle which you can access using \#1, but it sets the \langle key \rangle=(value) list inside of \langle set \rangle, so is equal to \texttt{\ekvset\{(set)\{(key)=(value), ...\} \ekvset\{(key)=(value), ...\} \ekvset\{(set)\{(key)=(value), ...\}.}

snmeta \langle key \rangle = \{(set)\{(key)=(value), ...\} 

And the last meta variant. snmeta is a combination of smeta and nmeta. It doesn’t take an argument and sets the \langle key \rangle=(value) list inside of \langle set \rangle.

set \langle key \rangle = \{(set)\} 

This will define \langle key \rangle to change the set of the current \texttt{\ekvset} invocation to \langle set \rangle. You can omit \langle set \rangle (including the equals sign), which is the same as using \texttt{set \langle key \rangle = \{(key)\}}. The created set key will not take a \langle value \rangle. Note that just like in \texttt{expX}, it’ll not be checked whether \langle set \rangle is defined and you’ll get a low-level \LaTeX{} error if you use an undefined \langle set \rangle.
choice  choice (key) = {⟨value⟩=⟨definition⟩, ...}  protected long

Defines ⟨key⟩ to be a choice key, meaning it will only accept a limited set of values. You should define each possible ⟨value⟩ inside of the ⟨value⟩=⟨definition⟩ list. If a defined ⟨value⟩ is passed to ⟨key⟩ the ⟨definition⟩ will be left in the input stream. You can make individual values protected inside the ⟨value⟩=⟨definition⟩ list. By default a choice key is expandable, an undefined ⟨value⟩ will throw an error in an expandable way.

1.3 Bugs

I don’t think there are any (but every developer says that), if you find some please let me know, either via the email address on the first page or on GitHub: https://github.com/Skillmon/tex_expkv-def

1.4 Example

The following is an example code defining each base key-type once. Please admire the very creative key-name examples.

\texttt{ekvdefinekeys \{example\}}
\{ 
  ,long code keyA = #1 
  ,noval keyA = NoVal given 
  ,bool keyB = \keyB 
  ,boolTF keyC = \keyC 
  ,store keyD = \keyD 
  ,data keyE = \keyE 
  ,dataT keyF = \keyF 
  ,int keyG = \keyG 
  ,dimen keyH = \keyH 
  ,skip keyI = \keyI 
  ,toks keyJ = \keyJ 
  ,default keyJ = \empty t e s t 
  ,box keyK = \keyK 
  ,qdefault keyK = text 
  ,choice keyL = 
  \{ 
    ,protected 1 = \texttt{ttt\{a\}} 
    ,2 = b 
    ,3 = c 
    ,4 = d 
    ,5 = e 
  \} 
  ,edefault keyL = 2 
  ,meta keyM = \{keyA=#1,keyB=false\} 
  ,data 
\}

Since the data type might be a bit strange, here is another usage example for it.
\ekvdefinekeys{ex}
\{
\, data name = \Pname
\, data age = \Page
\, dataT hobby = \Phobby
\}
\newcommand\Person [1]
\{
\begingroup
\ekvset{ex}{#1}
\begin{description}
\item \话box{\Pname}{ } \errmessage{A person requires a name}}
\item \话box{\Page}{textit{A person requires an age}}
\item \话box{\Phobby}{item \Hobbies}
\end{description}
\endgroup
\}
\Person{name=Jonathan P. Spratte, age=young, hobby=\TeX\ coding}
\Person{name=Some User, age=unknown, hobby=Reading Documentation}
\Person{name=Anybody, age=any}

In this example a person should have a name and an age, but doesn't have to have hobbies. The name will be displayed as the description item and the age in Italics. If a person has no hobbies the description item will be silently left out. The result of the above code looks like this:

Jonathan P. Spratte

Age young

Hobbies \TeX\ coding

Some User

Age unknown

Hobbies Reading Documentation

Anybody

Age any

1.5 License

Copyright © 2020 Jonathan P. Spratte

This work may be distributed and/or modified under the conditions of the \LaTeX Project Public License (LPPL), either version 1.3c of this license or (at your option) any later version. The latest version of this license is in the file:
http://www.latex-project.org/lppl.txt

This work is “maintained” (as per LPPL maintenance status) by Jonathan P. Spratte.
2 Implementation

2.1 The LaTeX Package

Just like for exPkv we provide a small LaTeX package that sets up things such that we behave nicely on LaTeX packages and files system. It’ll \input the generic code which implements the functionality.

\begin{verbatim}
\RequirePackage{expkv}
\def\ekvd@tmp
{%
  \ProvidesFile{expkv-def.tex} [%
    \ekvdDate\space v\ekvdVersion\space a key-defining frontend for expkv]%
%
}\input{expkv-def.tex}
\ProvidesPackage{expkv-def} [%
  \ekvdDate\space v\ekvdVersion\space a key-defining frontend for expkv]}
\end{verbatim}

2.2 The Generic Code

The rest of this implementation will be the generic code.

Load \texttt{exPkv} if the package didn’t already do so – since \texttt{exPkv} has safeguards against being loaded twice this does no harm and the overhead isn’t that big. Also we reuse some of the internals of \texttt{exPkv} to save us from retyping them.

\begin{verbatim}
\input exPkv
\end{verbatim}

We make sure that \texttt{expkv-def.tex} is only input once:

\begin{verbatim}
\expandafter\ifx\csname ekvdVersion\endcsname\relax
\else
\expandafter\endinput
\fi
\end{verbatim}

\texttt{\ekvdVersion} \texttt{\ekvdDate}

We’re on our first input, so lets store the version and date in a macro.

\begin{verbatim}
\def\ekvdVersion{0.3}
\def\ekvdDate{2020-04-29}
\end{verbatim}

(End definition for \texttt{\ekvdVersion} and \texttt{\ekvdDate}. These functions are documented on page 2.)

If the LaTeX format is loaded we want to be a good file and report back who we are, for this the package will have defined \texttt{\ekvd@tmp} to use \texttt{\ProvidesFile}, else this will expand to a \texttt{\relax} and do no harm.

\begin{verbatim}
\csname ekvd@tmp\endcsname
\end{verbatim}

Store the category code of \texttt{\relax} to later be able to reset it and change it to \texttt{11} for now.

\begin{verbatim}
\expandafter\chardef\csname ekvd@tmp\endcsname=\catcode"\relax
\catcode\relax=11
\end{verbatim}

\texttt{\ekvd@tmp} will be reused later to handle expansion during the key defining. But we don’t need it to ever store information long-term after \texttt{\expkv\def} was initialized.

\texttt{\expkv\def} will use \texttt{\ekvd@long} and \texttt{\ekvd@prot} to store whether a key should be defined as \texttt{\long} or \texttt{\protected}, and we have to clear them for every new key. By default they’ll just be empty.

\begin{verbatim}
\def\ekvd@empty{}
\protected\def\ekvd@clear@prefixes
{%
\end{verbatim}
\let\ekvd@long\ekvd@empty
\let\ekvd@prot\ekvd@empty
}
\ekvd@clear@prefixes

(End definition for \ekvd@long and others.)

\ekvdefinekeys This is the one front-facing macro which provides the interface to define keys. It’s using \ekvparseto handle the ⟨key⟩=⟨value⟩ list, the interpretation will be done by \ekv@noarg and \ekvd@. The ⟨set⟩ for which the keys should be defined is stored in \ekvd@set.
\protected\def\ekvdefinekeys#1%{
    \def\ekvd@set{#1}%
    \ekvparse\ekvd@noarg\ekvd@
}

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

\ekv@noarg \ekvd@noarg just places a special marker and gives control to \ekvd@. \ekvd@ has to test whether there is a space inside the key and if so calls the prefix grabbing routine, else we throw an error and ignore the key.
\protected\def\ekv@noarg#1\ekvd@noarg@mark\ekvd@
\protected\long\def\ekvd@#1#2%{
    \ekv@clear@prefixes
    \ekvd@ifspace{#1}\
        \ekvd@prefix\ekv@mark#1\ekv@stop{#2}\
    \ekvd@err@missing@prefix{#1}\
}

(End definition for \ekvd@noarg and \ekvd@)

\ekvd@prefix \ekvd@prefix@ \exp@p@\exp@n@er separates prefixes into two groups, the first being prefixes in the \TeX{} sense (long and protected) which use \TeX{}@@p@ in their name, the other being key-types (\code{}, \int{}, etc.) which use \TeX{}@@t@ instead. \ekvd@prefix splits at the first space and checks whether it’s a \TeX{}@@p@ or \TeX{}@@t@ type prefix. If it is neither throw an error and gobble the definition (the value).
\protected\def\ekvd@prefix#1\ekvd@prefix@\ekv@mark\ekvd@
\protected\def\ekvd@prefix@#1#2\ekv@stop\
    \ekv@ifdefined{ekvd@t@#1}\
        \csname ekvd@t@#1\endcsname{#2}\
    \ekv@ifdefined{ekvd@p@#1}\
        \csname ekvd@p@#1\endcsname{#2}\
    \ekvd@err@undefined@prefix{#1}\@gobble\
}

(End definition for \ekvd@prefix and \ekvd@prefix@)
The \texttt{@p@} type prefixes are all just modifying a following \texttt{@t@} type, so they will need to search for another prefix. This is true for all of them, so we use a macro to handle this. It'll throw an error if there is no other prefix.

\begin{verbatim}
\protected\def\ekvd@prefix@after@p#1\%
\{\ekvd@ifspace{#1}\%
{\ekvd@prefix#1\ekv@stop}\%
\else\expandafter\ekvd@err@missing@prefix\expandafter{\ekv@gobble@mark#1}\%
\@gobble\%
\}\%
\end{verbatim}

(End definition for \texttt{\ekvd@prefix@after@p}).

Define the \texttt{@p@} type prefixes, they all just store some information in a temporary macro and call \texttt{\ekvd@prefix@after@p}.

\begin{verbatim}
\protected\def\ekvd@p@long{\let\ekvd@long\long\ekvd@prefix@after@p}
\protected\def\ekvd@p@protected{\let\ekvd@prot\protected\ekvd@prefix@after@p}
\let\ekvd@p@protect\ekvd@p@protected
\end{verbatim}

(End definition for \texttt{\ekvd@p@long, \ekvd@p@protected, and \ekvd@p@protect}).

### 2.2.1 Key Types

\texttt{\ekvd@t@set} The \texttt{set} type is quite straightforward, just define a \texttt{NoVal} key to call \texttt{\ekvchangeset}.

\begin{verbatim}
\protected\def\ekvd@t@set#1#2\%
\{\ekvd@assert@not@long{set #1}\%
\ekvd@assert@not@protected{set #1}\%
\ekvd@ifnoarg{#2}\%
{\ekvdefNoVal\ekvd@set{#1}{\ekvchangeset{#1}}}\%
\else\ekv@ifempty{#2}\%
{\ekvd@err@missing@definition{set #1}}\%
\{\ekvdefNoVal\ekvd@set{#1}{\ekvchangeset{#2}}}\%
\}\%
\end{verbatim}

(End definition for \texttt{\ekvd@t@set}).

\texttt{\ekvd@type@noval} \texttt{\ekvd@t@noval} \texttt{\ekvd@t@enoval} Another pretty simple type, \texttt{noval} just needs to assert that there is a definition and that \texttt{long} wasn't specified. There are types where the difference in the variants is so small, that we define a common handler for them, those common handlers are named with \texttt{@type@}. \texttt{noval} and \texttt{enoval} are so similar that we can use such a \texttt{@type@} macro, even if we could've done \texttt{noval} in a slightly faster way without it.

\begin{verbatim}
\protected\long\def\ekvd@type@noval#1#2#3#4\%
\{\ekvd@assert@arg{#1noval #3}{#4}\%
\ekvd@assert@not@long{#1noval #3}\%
\ekvdefNoVal\ekvd@set{#1}{\ekvchangeset{#1}}\%
\}\%
\end{verbatim}

(End definition for \texttt{\ekvd@type@noval}).
\protect\def\ekvd@t@noval{\ekvd@type@noval}\def
\protect\def\ekvd@t@enoval{\ekvd@type@noval e\edef}

(End definition for \ekvd@type@noval, \ekvd@t@noval, and \ekvd@t@enoval.)

\ekvd@type@code
\ekvd@t@code
\ekvd@t@ecode
code is simple as well, ecode has to use \edef on a temporary macro, since \exPkp
doesn't provide an \ekvedef.

\protected\long\def\ekvd@type@code#1#2#3#4\%
{\
\ekvd@assert@arg{#1code #3}{#4}
{\%
 \ekvd@prot\ekvd@long#2\ekvd@tmp##1{#4}\%
 \ekvlet\ekvd@set{#3}\ekvd@tmp
 \}%
}\%
\protect\def\ekvd@t@code{\ekvd@type@code{}\def}
\protect\def\ekvd@t@ecode{\ekvd@type@code e\edef}

(End definition for \ekvd@type@code, \ekvd@t@code, and \ekvd@t@ecode.)

\ekvd@type@default
\ekvd@t@default
\ekvd@t@qdefault
\ekvd@type@default asserts there was an argument, also the key for which one wants to
set a default has to be already defined (this is not so important for default, but qdefault
requires is). If everything is good, \edef a temporary macro that expands \ekvd@set
and the \csname for the key, and in the case of qdefault does the first expansion step of
the key-macro.

\protected\long\def\ekvd@type@default#1#2#3#4\%
{\%
 \ekvd@assert@arg{#1default #3}{#4}
 {\%
 \ekvifdefined\ekvd@set{#3}\%
 {\%
 \ekvd@assert@not@long{#1default #3}\%
 \ekvd@prot\edef\ekvd@tmp
 {\%
 \unexpanded\expandafter\endcsname\ekvd@set{#3}\%
 \csname\ekv@name\ekvd@set{#3}\endcsname{#4}\%
 }%
 \ekvletNoVal\ekvd@set{#3}\ekvd@tmp
 }%
 \{\ekvd@err@undefined@key{#3}\%
 }%
 }
\protect\def\ekvd@t@default{\ekvd@type@default{}{}\}
\protect\def\ekvd@t@qdefault{\ekvd@type@default q{\expandafter\expandafter}}

(End definition for \ekvd@type@default, \ekvd@t@default, and \ekvd@t@qdefault.)

\ekvd@t@edefault
edefault is too different from default and qdefault to reuse the @type@ macro, as it
doesn't need \unexpanded inside of \edef.

\protected\long\def\ekvd@t@edefault#1#2\%
{\%
 \ekvd@assert@arg{edefault #1}{#2}\
}
\ekvifdefined\ekvd@set{#1}\%
  {%
    \ekvd@assert@not@long{edefault #1}\%
    \ekvd@prot@edef\ekvd@tmp
    \csname\ekv@name\ekvd@set{#1}\endcsname\ekvd@set{#1}\ekvd@tmp
  }%
  {\ekvletNoVal\ekvd@set{#1}\ekvd@tmp}
{\ekvd@err@undefined@key{#1}}%
}

(End definition for \ekvd@t@edefault.)

\ekvd@t@initial
\ekvd@t@oinitial
\ekvd@t@einitial
\long\def\ekvd@t@initial#1#2\%
  {%
    \ekvd@assert@arg{initial #1}{#2}\%
    \ekvifdefined\ekvd@set{#1}\%
      {%
        \ekvd@assert@not@long{initial #1}\%
        \ekvd@assert@not@protected{initial #1}\%
        \csname\ekv@name\ekvd@set{#1}\endcsname\ekvd@set{#1}\expandafter\endcsname\expandafter
      }%
      {\ekvd@err@undefined@key{#1}}%
  }%
\long\def\ekvd@t@oinitial#1#2\%
  {%
    \ekvd@assert@arg{oinitial #1}{#2}\%
    \ekvifdefined\ekvd@set{#1}\%
      {%
        \ekvd@assert@not@long{oinitial #1}\%
        \ekvd@assert@not@protected{oinitial #1}\%
        \csname\ekv@name\ekvd@set{#1}\endcsname\ekvd@set{#1}\expandafter\endcsname\expandafter\expandafter
      }%
      {\ekvd@err@undefined@key{#1}}%
  }%
\long\def\ekvd@t@einitial#1#2\%
  {%
    \ekvd@assert@arg{einitial #1}{#2}\%
    \ekvifdefined\ekvd@set{#1}\%
      {%
        \ekvd@assert@not@long{einitial #1}\%
        \ekvd@assert@not@protected{einitial #1}\%
        \edef\ekvd@tmp{#2}\%
        \csname\ekv@name\ekvd@set{#1}\endcsname\expandafter\endcsname\expandafter\expandafter
      }%
      {\ekvd@err@undefined@key{#1}}%
  }%
The boolean types are a quicker version of a choice that accept true and false, and set up the NoVal action to be identical to \(\langle \text{key}\rangle = \text{true}\). The true and false actions are always just \let\ting the macro in \#7 to some other macro (e.g., \iftrue).

\begin{verbatim}
\protected\def\ekvd@type@bool#1#2#3#4#5#6#7\%
  \ekvd@assert@filledarg{#1bool#2 #6}{#7}\%
  \ekvd@newlet#7#5\%
  \ekvd@type@choice{#1bool#2}{#6}\%
  \protected\expandafter\def\csname\ekvd@choice@name\ekvd@set{#6}{true}\endcsname{#3\let#7#4}\%
  \protected\expandafter\def\csname\ekvd@choice@name\ekvd@set{#6}{false}\endcsname{#3\let#7#5}\%
\end{verbatim}

\protected\def\ekvd@t@bool{T}\protectdef@bool{T}\protectdef@boolTF{T}\protectdef@boolTF{T}\% (End definition for \ekvd@type@bool and others.)

\begin{verbatim}
\protected\def\ekvd@type@data#1#2#3#4#5#6#7\%
  \ekvd@assert@filledarg{#1data#2 #6}{#7}\%
  \ekvd@newlet#7#3\%
  \ekvd@long\protectdef@data{T}\protectdef@dataT{T}\%
\end{verbatim}

\protected\def\ekvd@t@data\protectdef@data\%
\protected\def\ekvd@t@edata\protectdef@data e\%
\protected\def\ekvd@t@gdata\protectdef@data g\%
\protected\def\ekvd@t@xdata\protectdef@data x\%
\protected\def\ekvd@t@dataT\protectdef@data{T}\%
\protected\def\ekvd@t@edataT\protectdef@data e{T}\%
\protected\def\ekvd@t@gdataT\protectdef@data g{T}\%
\protected\def\ekvd@t@xdataT\protectdef@data x{T}\%

(End definition for \ekvd@type@data and others.)

\begin{verbatim}
\protected\def\ekvd@type@box\%
\end{verbatim}

Set up our boxes. Though we’re a generic package we want to be colour safe, so we put an additional grouping level inside the box contents, for the case that someone uses color.

\begin{verbatim}
\protected\def\ekvd@newreg\%
\end{verbatim}

\protected\def\ekvd@newreg\%

(End definition for \ekvd@type@box and others.)

\end{verbatim}
\ekvd@assert@filledarg{\#1 box \#3}{\#4}\
\ekvd@newreg\#4{box}\
\protected\ekvd@long\ekvdef\ekvd@set{\#3}\
\{\#2 setbox\#4 hbox{\begingroup\#1 endgroup}\}\%\
\}
\protected\def\ekvd@t@box{\ekvd@type@box{}{}}
\protected\def\ekvd@t@gbox{\ekvd@type@box g\global}
(End definition for \ekvd@type@box, \ekvd@t@box, and \ekvd@t@gbox.)

\ekvd@type@toks
\ekvd@t@toks
\ekvd@t@gtoks
Similar to box, but set the toks.
\ekvd@assert@filledarg{\#1 toks \#3}{\#4}\
\ekvd@newreg\#4{toks}\
\protected\ekvd@long\ekvdef\ekvd@set{\#3}{\#2\#4{##1}}\
\}
\protected\def\ekvd@t@toks{\ekvd@type@toks{}{}}
\protected\def\ekvd@t@gtoks{\ekvd@type@toks{g}\global}
(End definition for \ekvd@type@toks, \ekvd@t@toks, and \ekvd@t@gtoks.)

\ekvd@type@apptoks
\ekvd@t@apptoks
\ekvd@t@gapptoks
Just like toks, but expand the current contents of the toks register to append the new contents.
\ekvd@assert@filledarg{\#1 appptoks \#3}{\#4}\
\ekvd@newreg\#4{toks}\
\protected\ekvd@long\ekvdef\ekvd@set{\#3}{\#2\#4\expandafter{\the\#4##1}}\
\}
\protected\def\ekvd@t@apptoks{\ekvd@type@apptoks{}{}}
\protected\def\ekvd@t@gapptoks{\ekvd@type@apptoks{g}\global}
(End definition for \ekvd@type@apptoks, \ekvd@t@apptoks, and \ekvd@t@gapptoks.)

\ekvd@type@reg
\ekvd@t@int
\ekvd@t@eint
\ekvd@t@gint
\ekvd@t@xint
\ekvd@t@dimen
\ekvd@t@edimen
\ekvd@t@gdimen
\ekvd@t@xdimen
\ekvd@t@skip
\ekvd@t@eskip
\ekvd@t@gskip
\ekvd@t@xskip
The \ekvd@type@reg can handle all the types for which the assignment will just be \langle register\rangle=\langle value\rangle.
\ekvd@assert@filledarg{\#1 reg\#1\#2\#3\#4\#5\#6\#7}{\#8}\
\ekvd@newreg\#8{\#7}\
\protected\ekvd@long\ekvdef\ekvd@set{\#6}{\#3\#7=\#4\#1\#5\relax}\
\}
\protected\def\ekvd@t@int{\ekvd@type@reg{int}{\count}}{}
\protected\def\ekvd@t@eint{\ekvd@type@reg{eint}{\count}{\numexpr\relax}}
\protected\def\ekvd@t@gint{\ekvd@type@reg{gint}{\count}\global{}}
\protected\def\ekvd@t@xint{\ekvd@type@reg{xint}{\count}\global{\numexpr\relax}
The none-expanding store types use an `\edef` or `\xdef` and `\unexpanded` to be able to also store # easily.

```
\protected\long\def\ekvd@type@store#1#2#3#4\%{
  #1\assert@filledarg{#1store #3}{#4}\%
  \unless\ifdefined#4\let#4\ekvd@empty\fi
  \protected\ekvd@long\ekvdef\ekvd@set{#3}{#2#4{\unexpanded{##1}}}%
}\protect\def\ekvd@type@store{\ekvd@type@store}{\edef}
\protected\def\ekvd@type@store{\ekvd@type@store}{\xdef}
\protect\long\def\ekvd@type@store{\ekvd@type@store}{\edef}
\protect\def\ekvd@type@store{\ekvd@type@store}{\edef}
```

(End definition for `\ekvd@type@store`.)

And the straight forward estore types.

```
\protected\long\def\ekvd@type@estore#1#2#3#4\%{
  #1\assert@filledarg{#1store #3}{#4}\%
  \unless\ifdefined#4\let#4\ekvd@empty
  \protected\ekvd@long\ekvdef\ekvd@set{#3}{#2#4{#1}}%
  \protected\def\ekvd@type@estore{\ekvd@type@estore}{\edef}
\protect\def\ekvd@type@estore{\ekvd@type@estore}{\edef}
\protect\long\def\ekvd@type@estore{\ekvd@type@estore}{\edef}
```

(End definition for `\ekvd@type@estore`.)

```
\protected\long\def\ekvd@type@meta#1#2#3#4\%{
  #1\assert@filledarg{#1meta #3}{#4}\%
  \unless\ifdefined#4\let#4\ekvd@empty
  \protected\ekvd@long\ekvdef\ekvd@set{#3}{#2#4{#1}}%
  \protected\def\ekvd@type@meta#1#2#3#4\%{
    \edef\ekvd@tmp\ekvd@set{#1meta #4}{#5}%
    \edef\ekvd@tmp\expandafter{\ekvd@type@meta#1#2#3#4\%}{\ekvd@tmp}{#5}{#3}{\ekvd@set{#4}}
  }
\protect\def\ekvd@type@meta\%{
  \edef\ekvd@tmp\ekvd@set{#1meta #1}{#2}%
  \edef\ekvd@tmp\expandafter{\ekvd@type@meta#1#2}{\ekvd@set{#1}{#2}}%
```

(End definition for `\ekvd@type@meta`.)
\protected\def\ekvd@type@meta@b
  {\% \expandafter\ekvd@type@meta@c\expandafter}
\protected\long\def\ekvd@type@meta@c#1#2\%
  {\ekvd@prot\long\def\ekvd@tmp#2{#1}\%}
\protected\def\ekvd@type@meta{\ekvd@type@meta@b\ekvlet{##1}}
\protected\long\def\ekvd@t@meta#1#2\%
  {\ekvd@assert@not@long{nmeta \#1}\% \ekvd@type@meta n\ekvletNoVal{}{#1}{#2}\%}
\protected\long\def\ekvd@t@smeta#1#2#3#4#5\%
  {\ekvd@assert@twoargs{s#1meta \#4}{#5}\% \ekvd@type@meta@a#5{#3}\% #2\ekvd@set{\#4}\ekvd@tmp\%}
\protected\def\ekvd@t@smeta{\ekvd@type@smeta@\ekvlet{##1}}
\protected\long\def\ekvd@t@snmeta#1#2\%
  {\ekvd@assert@not@long{snmeta \#1}\% \ekvd@type@smeta n\ekvletNoVal{}{#1}{#2}\%}

(End definition for \ekvd@type@meta and others.)

\ekvd@type@meta smeta is pretty similar to meta, but needs two arguments inside of \langle value \rangle, such that the first is the \langle set \rangle for which the sub-\ekvset and the second is the \langle key \rangle=\langle value \rangle list.
\protected\long\def\ekvd@type@smeta#1#2#3#4#5\%
  {\ekvd@assert@twoargs{s#1meta \#4}{\#5}\% \ekvd@type@meta@#1\#2\#3\#4\#5\%}
\protected\def\ekvd@type@smeta{\ekvd@type@smeta@\ekvlet{##1}}
\protected\long\def\ekvd@t@smeta{\ekvd@type@smeta@\ekvlet{##1}}
\protected\long\def\ekvd@t@snmeta#1#2\%
  {\ekvd@assert@not@long{snmeta \#1}\% \ekvd@type@smeta n\ekvletNoVal{}{#1}{#2}\%}

(End definition for \ekvd@type@smeta and others.)

\ekvd@type@choice The choice type is by far the most complex type, as we have to run a sub-parser on the choice-definition list, which should support the @p@ type prefixes as well (but long will always throw an error, as they are not allowed to be long). \ekvd@type@choice will just define the choice-key, the handling of the choices definition will be done by \ekvd@populate@choice.
\protected\def\ekvd@type@choice#1#2\%
  {\% \ekvd@assert@not@long{#1 #2}\% \ekvd@prot\edef\ekvd@tmp{\#1}\%
    \unexpanded\{\ekvd@choice@name\ekvd@set{\#2}{\#1}\% \ekvlet\ekvd@set{\#2}\ekvd@tmp\%
\ekvd@populate@choice just uses \ekvparsen and then gives control to \ekvd@populate@choice@noarg, which throws an error, and \ekvd@populate@choice@.
\protected\def\ekvd@populate@choice

16
\ekvparse\ekvd@populate@choice@noarg\ekvd@populate@choice@
\protected\long\def\ekvd@populate@choice@noarg#1\protect\{
  \expandafter\ekvd@err@missing@definition\expandafter\{\ekvd@set@choice : #1\}\protect\}
\ekvd@populate@choice@

\ekvd@populate@choice@ runs the prefix-test, if there is none we can directly define the
choice, for that \ekvd@set@choice will expand to the current choice-key's name, which
will have been defined by \ekvd@set@choice. If there is a prefix run the prefix grabbing
routine, which was altered for \@type@choice.

\protected\long\def\ekvd@populate@choice@#1#2\protect\{
  \ekvd@clear@prefixes
  \expandafter\ekvd@assert@arg\expandafter\{\ekvd@set@choice : #1\}{#2}\protect\{
    \ekv@strip{#1}\ekvd@choice@prefix@\ekv@mark
  \}
  \expandafter\def
  \csname\ekvd@choice@name\ekvd@set\ekvd@set@choice{#1}\endcsname
  \protect\{#2\}
\protect\}
\protected\def\ekvd@choice@prefix#1\protect\{
  \ekv@strip{#1}\ekvd@choice@prefix@
\}
\protected\def\ekvd@choice@prefix@#1#2\ekv@stop\protect\{
  \ekv@ifdefined{ekvd@choice@p@#1}\protect\{
    \csname ekvd@choice@p@#1@endcsname
    \ekv@strip{#2}\{\ekvd@set@choice{#1}\endcsname
  \}
\protect\}
\else
  \ekvd@err@undefined@prefix{#1}\@gobble
\protect\}
\protect\}
\protected\def\ekvd@choice@p@protected\protect\{
  \let\ekvd@choice@p@protect\ekvd@choice@p@protected
\protected\def\ekvd@choice@p@long\protect\{
  \expandafter\ekvd@choice@p@long@\expandafter\{\ekv@gobble@mark#1\}
  \expandafter\ifspace{#1}\protect\{
    \expandafter\ekvd@choice@p@long@
  \}
\protect\}
\end{verbatim}

17
Finally we’re able to set up the \texttt{@t@choice} macro, which has to store the current choice-key’s name, define the key, and parse the available choices.

\begin{verbatim}
\protected\long\def\ekvd@t@choice#1#2{%
  \ekvd@assert@arg{choice #1}{#2}%
  \ekvd@type@choice{choice}{#1}%
  \def\ekvd@set@choice{#1}%
  \ekvd@populate@choice{#2}%
}
\end{verbatim}

(End definition for \texttt{\ekvd@type@choice} and others.)

\subsection*{2.2.2 Key Type Helpers}

There are some keys that might need helpers during their execution (not during their definition, which are gathered as \texttt{@type@} macros). These helpers are named \texttt{@h@}.

\begin{verbatim}
\def\ekvd@h@choice#1{%
  \expandafter\ekvd@h@choice@
  \csname\ifcsname#1\endcsname#1\else relax\fi\endcsname{#1}%
}
\def\ekvd@h@choice@#1#2{%
  \ifx#1\relax
  \ekvd@err@choice@invalid{#2}%
  \expandafter\@gobble
  \fi
  #1%
}
\end{verbatim}

(End definition for \texttt{\ekvd@h@choice} and \texttt{\ekvd@h@choice@}.

\subsection*{2.2.3 Tests}

\texttt{\ekvd@noarg@mark} This macro serves as a flag for the case that no \texttt{value} was specified for a key. As such it is not a test, but exists only for some tests.

\begin{verbatim}
\def\ekvd@noarg@mark{\ekvd@noarg@mark}
\end{verbatim}

(End definition for \texttt{\ekvd@noarg@mark}.)

\texttt{\ekvd@fi@firstoftwo} While we can reuse many of the internals of \texttt{exp@} the specific case for this branch wasn’t needed by \texttt{exp@} and hence isn’t defined. We’ll need it, so we define it.

\begin{verbatim}
\long\def\ekvd@fi@firstoftwo{\fi\@secondoftwo\#1\#2{\fi\#1}}
\end{verbatim}

18
These macros test whether a control sequence is defined, if it isn't they define it, either via \let or via the correct \new\langle reg\rangle.

\long\def\ekvd@newlet#1#2\{%  
\unless\ifdefined#1\let#1#2\fi  
\protect\def\ekvd@newlet#1#2\{%  
\unless\ifdefined#1\csname new#2\endcsname#1\fi  
\}

A test for exactly two tokens can be reduced for an empty-test after gobbling two tokens, in the case that there are fewer tokens than two in the argument, only macros will be gobbed that are needed for the true branch, which doesn't hurt, and if there are more this will not be empty.

\long\def\ekvd@assert@twoargs#1#2\{%  
\ekvd@ifnottwoargs{#2}\{\ekvd@err@missing@definition{#1}\}\}  
\long\def\ekvd@ifnottwoargs#1\{%  
\ekvd@ifempty@gtwo#1\ekv@ifempty@B\ekv@ifempty@false\ekv@ifempty@A\ekv@ifempty@B\@firstoftwo  
\}

The test for an argument is just an \ifx comparison with our noarg@mark.

\long\def\ekvd@assert@arg#1#2\{%  
\ekvd@ifnoarg{#2}\{\ekvd@err@missing@definition{#1}\}\}  
\long\def\ekvd@ifnoarg#1\{%  
\ifx\ekvd@noarg@mark#1\ekvd@fi@firstoftwo  
\}

\long\def\ekvd@assert@filledarg#1#2\{%  
\ekvd@ifnoarg@or@empty{#2}\{\ekvd@err@missing@definition{#1}\}\}  
\long\def\ekvd@ifnoarg@or@empty#1\{%  
\ifx\ekvd@noarg@mark#1\\ekvd@fi@firstoftwo  
\}

\long\def\ekvd@assert@arg#1#2\{%  
\ekvd@ifnoarg@or@empty{#2}\{\ekvd@err@missing@definition{#1}\}\}
Some key-types don’t want to be `\long` or `\protected`, so we provide macros to test this and throw an error, this could be silently ignored but now users will learn to not use unnecessary stuff which slows the compilation down.

Yet another test which can be reduced to an if-empty, this time by gobbling everything up to the first space.

2.2.4 Messages

Most messages of `expkvdef` are not expandable, since they only appear during key-definition, which is not expandable anyway.

The non-expandable error messages are boring, so here they are:
The expandable error messages use \ekvd@err, which is just like \ekv@err from \expkv or the way expl3 throws expandable error messages. It uses an undefined control sequence to start the error message. \ekvd@err@choice@invalid will have to use this mechanism to throw its message. Also we have to retrieve the name parts of the choice in an easy way, so we use parentheses of catcode 8 here, which should suffice in most cases to allow for a correct separation.

\def\ekvd@err@choice@invalid#1% {
  \ekvd@err@choice@invalid@#1\ekv@stop%
}\begingroup \catcode40=8 \catcode41=8 \@firstofone{\endgroup
\def\ekvd@choice@name#1#2#3% {
  ekvd#1(#2)#3%
}\def\ekvd@err@choice@invalid@ ekvd#1(#2)#3\ekv@stop% {
  \ekvd@err{invalid choice '#3' ('#2', set '#1')}%
}\begingroup
\edef\ekvd@err {
  \endgroup\unexpanded{\long\def\ekvd@err}##1%
}\unexpanded{\expandafter\ekv@err\@firstofone%
  \unexpanded{\expandafter{\csname ! expkv-def Error:\endcsname}##1.}}%
\unexpanded{\ekv@stop}%
}\endgroup
\def\ekvd@err
(End definition for \ekvd@err@choice@invalid and others.)

Now everything that's left is to reset the category code of \@.
\catcode`\@=\ekvd@tmp
Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.

A

apptoks .................................. 5
\def .................................. 200, 204
gdimen .................................. 4
gint .................................. 4
gskip .................................. 5
gstore .................................. 4
gtoks .................................. 5

B

bool .................................. 4
boolTF .................................. 4
box .................................. 5

C

choice .................................. 6
code .................................. 3

D

data .................................. 4
dataT .................................. 4
default .................................. 3
dimen .................................. 4

ecode .................................. 3
data .................................. 4
edefault .................................. 3
edimen .................................. 4
einitial .................................. 3
eint .................................. 4
\ekvchangeset .......................... 68, 72
\ekvDate .................................. 2, 5, 9, 15
\ekvdef .................................. 195, 211, 222, 232, 242, 262, 272
\ekvdefinekeys .......................... 2, 27
\ekvdefNoVal .......................... 68, 72, 176
\ekvVersion .................................. 2, 5, 9, 15
\ekvifdefined .......................... 100, 119, 133, 146, 150
\ekvlet .................................. 91, 208, 312, 325
\ekvletNoVal .......................... 81, 108, 124, 302, 316
\ekvparse .......................... 30, 329
\ekvset .................................. 288
enval .................................. 3
eskip .................................. 5
estore .................................. 4

G

gapptoks .................................. 5
gbool .................................. 4
gboolTF .................................. 4
gbox .................................. 5
gdata .................................. 4
gdataT .................................. 4

T\LaTeX\ and L\LaTeX\ \texttt{2e} commands:

\ekv@err@ .................................. 506
\ekv@gobble@mark .......................... 56, 373
\ekv@ifdefined .......................... 43, 46, 355
\ekv@ifempty .................................. 70, 446
\ekv@ifempty@ .................................. 424, 463
\ekv@ifempty@false .......................... 422, 424, 459, 463
\ekv@ifempty@false .......................... 424, 459, 458, 459
\ekv@mark .................................. 37, 40, 341, 351