The keyvaltable package*

Richard Gay
gay@mais.informatik.tu-darmstadt.de

March 18, 2016

1 Introduction

The main goal of the keyvaltable package is to offer means for typesetting tables

1. easily and yet still looking rather nicely
2. in a way that separates content from presentation and
3. with re-usable layout for tables of the same type.

For this purpose, the package essentially builds on two concepts:

table types: A table type is identified by a table name, ⟨*tname*⟩, and has a list of columns as well as further properties such as the background colors of rows. Each column also has a name, ⟨*cname*⟩, as well as further properties such as the heading of the column and the alignment of the column's content.

key-value rows: A table row is specified by a list of key-value pairs, where the keys are column names and the corresponding values are the content of the cell in this row in the respective column.

The the display of the tables, the keyvaltable package builds on the packages tabu, longtable, and booktabs.

2 Usage

We start with a basic usage example. An explanation of the involved macros follows afterwards.

```
\NewKeyValTable{Recipe}{
  amount:     align=r;
  ingredient: align=l;
  step:       align=X[l];
}
\begin{KeyValTable}{Recipe}
\Row{amount=150g, ingredient=ice cream,
     step=put into bowl}
\Row{amount= 50g, ingredient=cherries,
     step=heat up and add to bowl}
\end{KeyValTable}
```

amount	ingredient	step
150g	ice cream	put into bowl
50g	cherries	heat up and add to bowl

*This document corresponds to keyvaltable v0.1, dated 2016/03/13. The package is available online at http://www.ctan.org/pkg/keyvaltable and https://github.com/Ri-Ga/keyvaltable.

The example code first defines a new table type, `Recipe`, along with the columns that belong to this type. There are three columns (`amount`, `ingredient`, and `step`), whose specifications are separated with semicolons. After the separating `:`, for each column, the macro configures the column alignment using the `align` key. The alignments `r` (right) and `l` (left) are the standard `tabular` alignments; the `X[l]` alignment is provided by the `tabu` package (see the documentation there).

After the definition of the table type, the example creates a table of the newly defined type. For this, the example uses the `KeyValTable` environment and the `\Row` macro, once for each row. The parameter `Recipe` of the `KeyValTable` identifies the type of the table. Most notably, each row can now produced by a single macro in which the content of the individual cells can be specified by pairs such as `amount=150g`, which puts "150g" into the `amount` column of the respective row.

The example above already shows that producing a rather nice-looking table – including alternating row colors as well as horizontal rules – without further ado. How the keyvaltable package can be used in the general case and how its visual appearance can be customized is subject of the remainder of this section.

2.1 Table Type Definition

`\NewKeyValTable` The general form of defining a table type is `\NewKeyValTable{`⟨*tname*⟩`}{`⟨*colspecs*⟩`}`, where

- ⟨*tname*⟩ is the name of the table type and
- ⟨*colspecs*⟩ is a semicolon-separated list of individual column specifications.

Each column specification is of the form

⟨*colname*⟩`:` ⟨*property*⟩`=`⟨*value*⟩`,` ⟨*property*⟩`=`⟨*value*⟩`,`...

In such a specification, ⟨*colname*⟩ represents the name of the column. The ⟨*property*⟩`=`⟨*value*⟩ pairs configure certain properties of the column. The ⟨*property*⟩ can be one of the following:

`align`: This property specifies the alignment of content in the column. The ⟨*value*⟩ can be set to any column alignment understood by the `tabu` environment of the `tabu` package. This particularly includes `l`, `c`, `r`, `p`, and `X`. By default (i.e., if this property is not set explicitly), this property is set to `l`.

`default`: This property specifies the default value of a cell in this column, i.e., in case that a `\Row` does not provide content for the cell. By default (i.e., if unset for a column), this is an empty string.

`format`: This property specifies a formatting macro for content of the cell. By default, the formatting macro takes the content as is but puts a `\strut` before and after the content (to yield a better vertical spacing).

`head`: This property specifies the content of the column's header row. The default value for this property is the name of the column.

`hidden`: This property specifies whether a table column shall be displayed or not. The ⟨*value*⟩ for this property can be `true` (to display the cell; the default) or `false` (to not display the cell).

2.2 Typesetting Tables

The first possibility for typesetting a table using the keyvaltable package, is via the `KeyValTable` environment, which the example at the beginning of this section shows. The second possibility is described in Section 2.3.

`KeyValTable` The `KeyValTable[`⟨*options*⟩`]{`⟨*tname*⟩`}` environment creates a table of type ⟨*tname*⟩. The type ⟨*tname*⟩ must have been created using `\NewKeyValTable` before. The environment itself already produces a table with the columns specified for the table type, produces a header row and some horizontal lines, and sets up background colors of rows.

The ⟨*options*⟩ override default configurations, if provided, and must then be a comma-separated list of ⟨*property*⟩=⟨*value*⟩ pairs. The following ⟨*property*⟩ names are available:

`rowbg`: This property specifies the background colors of content rows. The format of the ⟨*value*⟩ for this property must be ⟨*oddcolor*⟩`..`⟨*evencolor*⟩.[1] The default is `white..black!10`, i.e., alternatingly white and light very gray.

`headbg`: This property specifies the background color of the head row. The ⟨*value*⟩ must be a single color specification that is understood by the xcolor package.[2] The default is `black!14`.

`\Row` A table row is produced by the `\Row{`⟨*content*⟩`}` macro. The ⟨*content*⟩ must be a comma-separated list of ⟨*cname*⟩=⟨*text*⟩ pairs. The ⟨*cname*⟩ identifies a column that was registered for the table type ⟨*tname*⟩. The ⟨*text*⟩ specifies the content of the cell in the respective column. Each column for which no ⟨*text*⟩ is provided in ⟨*content*⟩, will result in a cell that is filled with the column's default value.

2.3 Tables of Collected Rows

As an alternative to producing a table within a single environment, the keyvaltable package offers a way to scatter individual rows throughout a document and display the full table later. This method can be useful when table rows are strongly connected to portions of text outside of the table. The method then allows specifying the rows together with the connected text rather than separately in the table environment. Table types for this method are defined via `\NewKeyValTable` as previously described.

`\AddKeyValRow` A table row is produced by the `\AddKeyValRow{`⟨*tname*⟩`}{`⟨*content*⟩`}` macro. The ⟨*tname*⟩ identifies the table type and the ⟨*content*⟩ provides the content of the cells in the row. The format of the ⟨*content*⟩ is the same as for the `\Row` macro described in Section 2.2.

`\ShowKeyValTable` A table of all the rows defined via `\AddKeyValRow` can be displayed by the

[1] The ⟨*value*⟩ is passed directly to `\taburowcolors` of the tabu package.
[2] The ⟨*value*⟩ is passed directly to the `\rowcolor` macro.

`\ShowKeyValTable[`⟨*options*⟩`]{`⟨*tname*⟩`}` macro. The parameters have the same meaning as for the `KeyValTable` environment. This macro resets the list of rows for the specified table type.

`KeyValTableContent` For simplifying the addition of rows, the `KeyValTableContent{`⟨*tname*⟩`}` environment can be used. In this environment, the `\Row` macro can be used just like in the `KeyValTable` environment. The only difference is that the `KeyValTableContent` environment does not cause the table to be displayed. For this, the `\ShowKeyValTable` macro can be used.

The following example demonstrates the use, based on the previously defined `Recipe` table type.

```
\AddKeyValRow{Recipe}{amount=3,
     ingredient=balls of snow,
     step=staple all 3 balls}
\begin{KeyValTableContent}{Recipe}
\Row{amount=1, ingredient=carrot,
     step=stick into top ball}
\Row{amount=2, ingredient=coffee beans,
     step=put diagonally above carrot}
\end{KeyValTableContent}
\ShowKeyValTable{Recipe}
```

amount	ingredient	step
3	balls of snow	staple all 3 balls
1	carrot	stick into top ball
2	coffee beans	put diagonally above carrot

2.4 Setting Global Defaults

`\kvtSet` The keyvaltable package allows changing the default values globally for the parameters of tables and columns. This can be done by using the `\kvtSet{`⟨*options*⟩`}` macro.

```
\kvtSet{headbg=red,default=?,align=r}
\NewKeyValTable{Defaults}{x; y}
\begin{KeyValTable}{Defaults}
\Row{x=1}
\Row{y=4}
\end{KeyValTable}
```

x	y
1	?
?	4

Notice the use of the `\NewKeyValTable` in the example. Column properties, including the separating `:` can be omitted completely, making the definition of a table type very simple.

2.5 Row Numbering

The mechanism of default column values enables a simple means for automatic row numbering. For this, one can use one of three row counters provided by the keyvaltable package: `kvtRow`, `kvtTypeRow`, and `kvtTotalRow`. The counters are explained after the following example, which demonstrates the use for the case of the `kvtRow` counter.

```
\NewKeyValTable{Numbered1}{
  line: align=r, head=\#,
        default=\arabic{kvtRow};
  text: align=l, head=\textbf{Text}}
\begin{KeyValTable}{Numbered1}
\Row{text=First row}
\Row{text=Second row}
\end{KeyValTable}
```

#	Text
1	First row
2	Second row

`kvtRow` The `kvtRow` counter counts the row in the *current* table. The row number excludes the header row of the table. If the table spans multiple pages, the row number also excludes the repeated headings on subsequent pages.

`kvtTypeRow` The `kvtTypeRow` counter counts the rows in the current table and includes the number of rows of all previous tables of the same type.

`kvtTotalRow` The `kvtTotalRow` counter counts the rows in the current table and includes the number of rows of all previous tables produced using the keyvaltable package.

3 Use with Other Packages

3.1 Computational Cells

The mechanism of cell formatting macros enables a simple means for automatically computing formulas contained in a column. This can be done, for instance using the xint package and a custom format macro (here `\Math`) that takes over the computation.

```
\usepackage{xintexpr}
\newcommand\Math[1]{%
   \xinttheexpr trunc(#1, 1)\relax}
\NewKeyValTable{Calculating}{
   type; value: align=r,format=\Math}
\begin{KeyValTable}{Calculating}
\Row{type=simple,   value=10+5.5}
\Row{type=advanced, value=0.2*(9+2^8)}
\end{KeyValTable}
```

type	value
simple	15.5
advanced	53.0

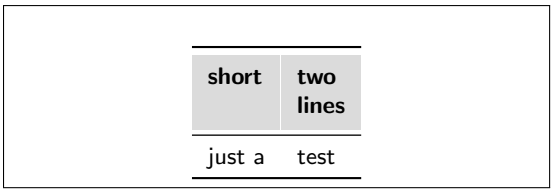
3.2 Cell Formatting

The keyvaltable package can be used together with the makecell package in at least two ways:

1. formatting header cells using the `head` property of columns;
2. formatting content cells using the `format` property of columns.

The following example gives an impression.

```
\usepackage{makecell}
\renewcommand\theadfont{\bfseries}
\renewcommand\theadalign{lt}
\NewKeyValTable{Header}{
   first:  head=\thead{short};
   second: head=\thead{two\\ lines};}
\begin{KeyValTable}{Header}
\Row{first=just a, second=test}
\end{KeyValTable}
```



short	two lines
just a	test

4 Related Packages

I'm not aware of any LaTeX packages that pursue similar goals or provide similar functionality. The following LaTeX packages provide loosely related functionalities to the keyvaltable package.

ctable: This package focuses on typesetting tables with captions and notes. With this package, the specification of table content is quite close to normal `tabular` environments, except that the package's table creation is done via a macro, `\ctable`.

easytable: This package provides an environment `TAB` which simplifies the creation of tables with particular horizontal and vertical cell alignments, rules around cells, and cell width distributions. In that sense, the package aims at simpler table creation, like the keyvaltable. However, the package does not pursue separation of content from presentation or re-use of table layouts.

tabularkv: Despite the similarity in the name, this package pursues a different purpose. Namely, this package provides means for specifying table options such as width and height through an optional key-value argument to the `tabularkv` environment. This package does not use a key-value like specification for the content of tables.

5 Future Work

- configurable default heading format
- possibility to use `\label` for assigning labels to row numbers (via `\refstepcounter` in final pass of tabu?)
- configurable table environments, at least between `longtabu` (the default) and `tabu`, but possibly also `tabular`, `tabularx`, and others; this could be realized via a `type` option for `\NewKeyValTable`
- improved row coloring that makes sure that the alternation re-starts on continued pages of a table that spans several pages

6 Implementation

We use etoolbox for some convenience macros that make the code more easily maintainable and use xkeyval for options in key–value form.

```
\RequirePackage{etoolbox}
\RequirePackage{xkeyval}
```

We use tabu for creating the tables and longtable for tables that can span multiple pages (via `longtabu`). We use booktabs for nice horizontal lines and xcolor for row coloring

```
\RequirePackage[table]{xcolor}
\RequirePackage{booktabs}
\RequirePackage{longtable,tabu}
```

6.1 Setting Defaults

`\kvtSet` The `\kvtSet{`⟨*options*⟩`}` set the default options, which apply to all tables typeset with the package.

```
\newcommand\kvtSet[1]{\bgroup
  \def\kvt@@presetqueue{\egroup}
  \setkeys[kvt]{defaults}{#1}{}%
  \kvt@@presetqueue}
```

`\kvt@lazypreset` The `\kvt@@lazypreset{`⟨*family*⟩`}{`⟨*head keys*⟩`}` macro collects a request collects a request for pre-setting ⟨*head keys*⟩ in family key ⟨*family*⟩. Using this macro, one can avoid causing problems with using xkeyval's `\presetkeys` inside the ⟨*function*⟩ defined for a key (e.g., via `\define@key`). The collected requests can be performed by expanding the `\kvt@@presetqueue` macro.

```
\newcommand\kvt@lazypreset[2]{%
  \appto\kvt@@presetqueue{\presetkeys[kvt]{#1}{#2}{}}}
```

`\mkv@dossvlist` The `\mkv@dossvlist{`⟨*list*⟩`}` macro parses a semicolon-separated list and runs `\do`⟨*item*⟩ for every element of the list.

```
\DeclareListParser{\mkv@dossvlist}{;}
```

`\kvt@addtableprop` The `\kvt@addtableprop{`⟨*name*⟩`}{`⟨*default*⟩`}` macro adds a new table option, named ⟨*name*⟩ and with default value ⟨*default*⟩.

```
\newcommand\kvt@addtableprop[2]{%
  \define@key[kvt]{defaults}{#1}{%
    \kvt@lazypreset{Table}{#1=##1}}%
  \presetkeys[kvt]{defaults}{#1=#2}{}%
  \define@cmdkey[kvt]{Table}{#1}{}%
  \presetkeys[kvt]{Table}{#1=#2}{}%
}
```

`\kvt@addcolumnprop` The `\kvt@addcolumnprop{`⟨*name*⟩`}{`⟨*default*⟩`}` macro adds a new column option, named ⟨*name*⟩ and with default value ⟨*default*⟩.

```
\newcommand\kvt@addcolumnprop[2]{%
```

The following makes the ⟨*name*⟩ available as an option for `\kvtSet` for setting a global default value to this column option.

```
\define@key[kvt]{defaults}{#1}{%
  \kvt@lazypreset{Column}{#1=##1}}%
\presetkeys[kvt]{defaults}{#1=#2}{}%
```

The following makes the ⟨*name*⟩ available as an option for the ⟨*colspecs*⟩ of `\NewKeyValTable` for setting a default value to the particular column.

```
\define@key[kvt]{Column}{#1}{%
  \csdef{kvt@col@#1@\kvt@@column}{##1}}%
\presetkeys[kvt]{Column}{#1=#2}{}%
}
```

`\kvt@addchoicecolumnprop` The `\kvt@addchoicecolumnprop{`⟨*name*⟩`}{`⟨*default*⟩`}{`⟨*choice*⟩`}` macro adds a new column option, named ⟨*name*⟩ and with default value ⟨*default*⟩.

```
\newcommand\kvt@addchoicecolumnprop[3]{%
```

The following makes the ⟨*name*⟩ available as an option for `\kvtSet` for setting a global default value to this column option.

```
\define@choicekey[kvt]{defaults}{#1}{#3}{%
  \kvt@lazypreset{Column}{#1=##1}}%
\presetkeys[kvt]{defaults}{#1=#2}{}%
```

The following makes the ⟨*name*⟩ available as an option for the ⟨*colspecs*⟩ of `\NewKeyValTable` for setting a default value to the particular column.

```
\define@choicekey[kvt]{Column}{#1}{#3}%
  {\csdef{kvt@col@#1@\kvt@@column}{##1}}%
\presetkeys[kvt]{Column}{#1=#2}{}%
}
```

The following are the known column properties and their defaults as well as the known table properties and their defaults.

```
\kvt@addtableprop{rowbg}{white..black!10}
\kvt@addtableprop{headbg}{black!14}
\kvt@addcolumnprop{default}{}
\kvt@addcolumnprop{format}{\kvt@struttedcell}
\kvt@addcolumnprop{align}{l}
\kvt@addcolumnprop{head}{}
\kvt@addchoicecolumnprop{hidden}{false}{false,true}
\kvtSet{}
```

`\kvt@struttedcell` The `\kvt@struttedcell{`⟨*arg*⟩`}` macro prefixes and suffixes the argument ⟨*arg*⟩ with a `\strut`. When used for formatting cell content, this makes sure that there is some vertical space between the content of a cell and the top and bottom of the row.

```
\newcommand\kvt@struttedcell[1]{\strut #1\strut}
```

6.2 Declaring Key-Value Tables

`\NewKeyValTable` The `\NewKeyValTable{`⟨*tname*⟩`}{`⟨*colspecs*⟩`}` declares a new key-value table type, identified by the given ⟨*tname*⟩. The columns of the table type are specified by ⟨*colspecs*⟩.

```
\newcommand\NewKeyValTable[2]{%
```

First initialize the "variables".

```
  \csdef{kvt@headings@#1}{}%
  \csdef{kvt@alignments@#1}{}%
  \csdef{kvt@colkeys@#1}{}%
  \csdef{kvt@rowcount@#1}{1}%
  \csdef{kvt@rows@#1}{}%
  \listadd\kvt@alltables{#1}%
```

Now parse ⟨*colspecs*⟩, a semicolon-separated list of individual column specifications, and add the columns to the table. Each `\do{`⟨*colspec*⟩`}` takes the specification for a single column.

```
  \def\do##1{%
    \kvt@parsecolspec{#1}##1::\@undefined}%
  \mkv@dossvlist{#2}%
}
```

The `\kvt@parsecolspec{`⟨*tname*⟩`}`⟨*cname*⟩`:`⟨*config*⟩`:`⟨*empty*⟩`\@undefined` takes a configuration ⟨*config*⟩ for a column ⟨*cname*⟩ in table ⟨*tname*⟩ and adds the column with the configuration to the table.

```
\def\kvt@parsecolspec#1#2:#3:#4\@undefined{%
  \def\kvt@@column{#1@#2}%
  \setkeys[kvt]{Column}{#3}%
```

The following stores the column's properties. The column is only added if the `hidden` option is not set to `true`.

```
  \ifcsstring{kvt@col@hidden@#1@#2}{true}{}{%
    \cseappto{kvt@alignments@#1}{\csexpandonce{kvt@col@align@#1@#2}}%
    \ifcsvoid{kvt@headings@#1}{}{\csappto{kvt@headings@#1}{&}}%
    \ifcsstring{kvt@col@head@#1@#2}{}%
      {\cseappto{kvt@headings@#1}{#2}}%
      {\cseappto{kvt@headings@#1}{\csexpandonce{kvt@col@head@#1@#2}}}%
    \listcsadd{kvt@colkeys@#1}{#2}%
  }%
```

The following creates the column key that can be used by the row macros to set the content of the column's content in that row.

```
  \define@cmdkey[KeyValTable]{#1}{#2}[]{}%
  \presetkeys[KeyValTable]{#1}{#2}{}%
}
```

`\kvt@alltables` The `\kvt@alltables` is an etoolbox list containing the names of all tables declared by `\NewKeyValTable`.

```
\newcommand\kvt@alltables{}
```

6.3 Row Numbering

The following counters simplify row numbering in key-value tables. One can use a table-local counter (`kvtRow`), a table-type local counter (`kvtTypeRow`), and a global counter (`kvtTotalRow`).

`kvtRow` The `kvtRow` counter can be used by cells to get the current row number. This row number (in contrast to `taburow`) does not count table headers. That is, `kvtRow` provides the current *content* row number, even in tables that are spread over multiple pages.

```
\newcounter{kvtRow}
```

`kvtTypeRow` The `kvtTypeRow` counter can be used by cells to get the current row number, including all previous rows of tables of the same type. This counter works together with the `\kvt@rowcount@`⟨*tname*⟩ macro, which keeps track of the individual row counts of the ⟨*tname*⟩ type.

```
\newcounter{kvtTypeRow}
```

`kvtTotalRow` The `kvtTotalRow` counter can be used by cells to get the current row number, including all previous `KeyValTable` tables.

```
\newcounter{kvtTotalRow}
\setcounter{kvtTotalRow}{1}
```

6.4 Key-Value Table Content

`KeyValTable` The `KeyValTable[`⟨*options*⟩`]{`⟨*tname*⟩`}` environment encloses a new table whose type is identified by the given ⟨*tname*⟩. Table options can be overridden by providing ⟨*options*⟩.

```
\newenvironment{KeyValTable}[2][]{%
  \bgroup%
  \def\Row##1{\kvt@AddKeyValRow{#2}{##1}\kvt@@row\\}%
  \kvt@StartTable{#2}{#1}%
  }{%
  \bottomrule%
  \end{longtabu}\egroup}
```

The following saves the row counter value outside the table environment but still in the then-local scope.

```
\AfterEndEnvironment{KeyValTable}{%
  \csdef{kvt@rowcount@\kvt@@recenttable}{\thekvtTypeRow}}
```

`\kvt@StartTable` The `\kvt@StartTable{`⟨*tname*⟩`}{`⟨*options*⟩`}` begins the table environment of a `KeyValTable`, displays the head row, and sets the row counters.

```
\newcommand\kvt@StartTable[2]{%
  \setkeys[kvt]{Table}{#2}%
```

The `\kvt@@recenttable` allows the `\AfterEndEnvironment` hook for `KeyValTable` to access the most recent table type.

```
  \gdef\kvt@@recenttable{#1}%
  \bgroup\edef\kvt@@do{\egroup
    \noexpand\taburowcolors[2] 2{\cmdkvt@Table@rowbg}%
    \noexpand\begin{longtabu}{\csuse{kvt@alignments@#1}}%
    \noexpand\toprule
    \noexpand\rowcolor{\cmdkvt@Table@headbg}%
  }\kvt@@do%
  \csuse{kvt@headings@#1}\\\midrule\endhead
  \setcounter{kvtRow}{1}%
  \setcounter{kvtTypeRow}{\csuse{kvt@rowcount@#1}}%
  \everyrow{%
    \addtocounter{kvtRow}{1}%
    \addtocounter{kvtTypeRow}{1}%
    \addtocounter{kvtTotalRow}{1}%
  }%
}
```

`\kvt@AddKeyValRow` The `\kvt@AddKeyValRow{`⟨*tname*⟩`}{`⟨*content*⟩`}` adds a new row to the current table of type ⟨*tname*⟩. The ⟨*content*⟩ is a key-value list that specifies the content of the individual cells in the row. The macro assumes that the current content of the table is stored in `\kvt@@row` and modifies this macro by appending the additional row.

```
\newcommand\kvt@AddKeyValRow[2]{%
  \setkeys[KeyValTable]{#1}{#2}%
```

The following loop uses `\do{`⟨*cname*⟩`}` to append the content of all columns (in the given format and using the given default value), where each column value is in `\cmdKeyValTable@`⟨*tname*⟩`@`⟨*cname*⟩. *NOTE:* Currently, the default value is formatted using the given format macro.

```
  \def\do##1{%
    \ifdefvoid\kvt@@row{}{\appto\kvt@@row{&}}%
    \eappto\kvt@@row{%
      \csexpandonce{kvt@col@format@#1@##1}{%
      \ifcsvoid{cmdKeyValTable@#1@##1}%
        {\csexpandonce{kvt@col@default@#1@##1}}%
        {\csexpandonce{cmdKeyValTable@#1@##1}}}%
    }%
  }\dolistcsloop{kvt@colkeys@#1}%
}
```

6.5 Collecting Key-Value Table Content

`\ShowKeyValTable` The `\ShowKeyValTable[`⟨*options*⟩`]{`⟨*tname*⟩`}` macro shows a table of type ⟨*tname*⟩ with given ⟨*options*⟩. The rows must have been collected using `\Row` in `KeyValTableContent` environments or using `\AddKeyValRow`.

```
\newcommand\ShowKeyValTable[2][]{%
  \bgroup
  \kvt@StartTable{#2}{#1}%
  \csuse{kvt@rows@#2}%
```

```
118   \bottomrule
119   \end{longtabu}%
120   \egroup
121   \csdef{kvt@rows@#2}{}}
```

`\AddKeyValRow` The `\AddKeyValRow{`⟨*tname*⟩`}{`⟨*content*⟩`}` adds a row with a given ⟨*content*⟩ to the existing content for the next table of type ⟨*tname*⟩. That is displayed with `\ShowKeyValTable`. The ⟨*content*⟩, like with `\kvt@AddKeyValRow`, is supposed to be a key-value list specifying the content of the cells in the row.

```
122 \newcommand\AddKeyValRow[2]{%
123   \bgroup%
124   \kvt@AddKeyValRow{#1}{#2}%
125   \csxappto{kvt@rows@#1}{\expandonce{\kvt@@row}\noexpand\\}%
126   \egroup}
```

`KeyValTableContent` The `KeyValTableContent{`⟨*tname*⟩`}` environment acts as a container in which rows can be specified without automatically being displayed. In this environment, rows can be specified via the `\Row{`⟨*content*⟩`}` macro, which is supposedly shorter than using `\AddKeyValRow`⟨*tname*⟩⟨*content*⟩.

```
127 \newenvironment{KeyValTableContent}[1]{%
128   \def\Row{\AddKeyValRow{#1}}}{}%
```

Change History

v0.1

General: Initial version 1

Index

Symbols

\@undefined 53, 56
\\ 77, 93, 125

A

\AddKeyValRow *3*, 122, 128
\addtocounter 97, 98, 99
\AfterEndEnvironment 82
\appto 11, 105

B

\begin 89
\bgroup 6, 76, 87, 115, 123
\bottomrule 80, 118

C

\cmdkvt@Table@headbg 91
\cmdkvt@Table@rowbg 88
\csappto 61
\csdef . 25, 33, 46, 47, 48, 49, 50, 83, 121
\cseappto 60, 63, 64
\csexpandonce . . 60, 64, 107, 109, 110
\csuse 89, 93, 95, 117
\csxappto 125

D

\DeclareListParser 12
\define@choicekey 29, 32

\define@cmdkey 17, 67
\define@key 14, 21, 24
\do 52, 104
\dolistcsloop 112

E

\eappto 106
\egroup 7, 81, 87, 120, 126
\end 81, 119
\endhead 93
environments:
KeyValTable *3*, 75
KeyValTableContent . *4*, 127
\everyrow 96
\expandonce 125

I

\ifcsstring 59, 62
\ifcsvoid 61, 108
\ifdefvoid 105

K

KeyValTable (environment) *3*, 75
KeyValTableContent (environment) *4*, 127
\kvt@@column 25, 33, 57
\kvt@@do 87, 92
\kvt@@presetqueue 7, 9, 11
\kvt@@recenttable 83, 86
\kvt@@row 77, 105, 106, 125
\kvt@addchoicecolumnprop 28, 42
\kvt@addcolumnprop . 20, 38, 39, 40, 41
\kvt@AddKeyValRow . 77, 102, 124
\kvt@addtableprop . . . 13, 36, 37
\kvt@alltables 51, 70
\kvt@lazypreset . . 10, 15, 22, 30
\kvt@parsecolspec 53, 56
\kvt@StartTable 78, 84, 116
\kvt@struttedcell 39, 44
\kvtRow *5*, 71
\kvtSet *4*, 6, 43
\kvtTotalRow *5*, 73
\kvtTypeRow *5*, 72

L

\listadd 51
\listcsadd 65

M

\midrule 93
\mkv@dossvlist 12, 54

N

\newcounter 71, 72, 73
\NewKeyValTable *2*, 45
\noexpand . . . 88, 89, 90, 91, 125

P

\presetkeys 11, 16, 18, 23, 26, 31, 34, 68

R

\RequirePackage . . . 1, 2, 3, 4, 5
\Row *3*, 77, 128
\rowcolor 91

S

\setcounter 74, 94, 95
\setkeys 8, 58, 85, 103
\ShowKeyValTable *3*, 114
\strut 44

T

\taburowcolors 88
\thekvtTypeRow 83
\toprule 90