The keyvaltable package*

Richard Grewe
r-g+tex@posteo.net

March 17, 2019

Abstract

The keyvaltable package's main goal is to facilitate typesetting tables…

(a)	…easily and yet still looking rather nicely	through horizontal rules and alternating row background colors by default;
(b)	…in a way that separates content from presentation	by table rows that are specified as lists 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;
(c)	…with re-usable layout for tables of the same type	through named table types, of which each has a list of columns as well as further properties such as the background colors of rows; each column, in turn, has a name as well as further properties such as the heading of the column and the alignment of the column's content.

Contents

*This document corresponds to keyvaltable v1.0, dated 2019/03/17. The package is available online at http://www.ctan.org/pkg/keyvaltable and https://github.com/Ri-Ga/keyvaltable.

1 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

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), which is used by default for creating the tables.

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.

1.1 Table Type Definition

`\NewKeyValTable[`⟨*options*⟩`]{`⟨*tname*⟩`}{`⟨*colspecs*⟩`}[`⟨*headers*⟩`]`

Table types are defined via the `\NewKeyValTable` macro, where

- ⟨*tname*⟩ is the name of the table type,
- ⟨*colspecs*⟩ is a semicolon-separated list of individual column specifications, and
- ⟨*options*⟩, if provided, specify table type options that override the default table options; they must then be a comma-separated list of ⟨*property*⟩=⟨*value*⟩ pairs; the list of table options can be found at the introduction of the `KeyValTable` environment on page 4.
- ⟨*headers*⟩, if provided, specifies custom table header rows. This argument is further described in Section 1.6. If this argument is omitted, a single header

row is produced (unless `showhead=false` is provided as an option) and the individual headers in this row are determined by ⟨*colspec*⟩.

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:

Key	Description and Possible Values	Default
`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`.	`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.	*(empty)*
`format`	This property specifies a formatting macro for content of the cell. The macro can take one argument and is provided with the content of the cell as its argument. 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).	`\kvtStrutted`
`head`	This property specifies the content of the column's header row. The default value for this property is the name of the column.	⟨*colname*⟩
`hidden`	This property specifies whether a table column shall be displayed or not. The ⟨*value*⟩ for this property can be `true` (to hide the cell; the default) or `false` (to display the cell).	`false`

1.2 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 1.3.

`\begin{KeyValTable}[`⟨*options*⟩`]{`⟨*tname*⟩`}`
`\end{KeyValTable}`

The `KeyValTable` 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:

Key	Description and Possible Values	Default
`shape`	This property specifies the table's shape. For ⟨*value*⟩, the package currently supports `multipage` and `onepage` as well as `tabular`, `tabularx`, and `longtable`. In case of `multipage`, the table may span multiple pages and on each page, the column header is repeated. In case of `onepage`, the table does not split into multiple pages. The remaining three values use the respective environment for producing the table (see Section 1.7 for the effect).	`multipage`
`width`	This property specifies the width of the table, if the selected `shape` supports it (see Section 1.7).	`\linewidth`
`showhead`	This property specifies whether the head row shall be shown. The ⟨*value*⟩ must be a Boolean (i.e., `true` or `false`), where `true` specifies that the head row is shown and `false` specifies that the head row is not shown.	`true`
`showrules`	This property specifies whether top and bottom rules as well as a rule below the head row are drawn (`true`) or not (`false`).	`true`
`headalign`	This property specifies the alignment for header cells. If left empty, each header cell receives the same alignment as the respective column.	*(empty)*
`headfmt`	This property specifies a format to be applied to all header cells. By default, the property is empty, meaning that header cells are formatted. Otherwise, the code provided as value to this key is prepended to the text of the header cells.	*(empty)*
`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. The ⟨*value*⟩ is passed directly to the `\rowcolor` macro.	`black!14`

```
\NewKeyValTable[shape=onepage,
   showhead=false,
   rowbg=blue!10..blue!15,
 ]{TabOptions}{
   opt: align=l, format=\texttt;
   val: align=l, format=\texttt;}
\begin{table}\centering
\begin{KeyValTable}{TabOptions}
\Row{opt=shape, val=onepage}
\Row{opt=showhead, val=false}
\Row{opt=rowbg, val=blue!10..blue!15}
\end{KeyValTable}
\caption{table options demo}
\end{table}
```

shape	onepage
showhead	false
rowbg	blue!10..blue!15

Table 4: table options demo

```
\NewKeyValTable[showrules=false,headbg=blue!25,
   headalign=c,headfmt=\bfseries,
 ]{TabOptions2}{
   opt: align=l, format=\texttt;
   val: align=l, format=\texttt;}
\begin{KeyValTable}{TabOptions2}
\Row{opt=showrules, val=false}
\Row{opt=headbg, val=blue!25}
\Row{opt=headalign, val=c}
\Row{opt=headfmt, val=\string\bfseries}
\end{KeyValTable}
```

opt	val
showrules	false
headbg	blue!25
headalign	c
headfmt	\bfseries

Figure 1: Examples for table options

Key	Description and Possible Values	Default
rowbg	This property specifies the background colors of content rows. The format of the ⟨*value*⟩ for this property must be ⟨*oddcolor*⟩..⟨*evencolor*⟩. The first row after the header is colored with ⟨*oddcolor*⟩, the second row with ⟨*evencolor*⟩, and so forth. Both colors must be understood by the xcolor package.	white..black!10

Figure 1 demonstrates the ⟨*options*⟩ in examples.

`\Row[`⟨*options*⟩`]{`⟨*content*⟩`}`

A table row is produced by the `\Row` 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.

The ⟨*options*⟩ argument customizes row properties and is further explained in Section 1.8.

1.3 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{`⟨*tname*⟩`}{`⟨*content*⟩`}`

A table row is produced by the `\AddKeyValRow` 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 1.2.

`\ShowKeyValTable[`⟨*options*⟩`]{`⟨*tname*⟩`}`

A table of all the rows defined via `\AddKeyValRow` can be displayed by the `\ShowKeyValTable` macro. The parameters have the same meaning as for the `KeyValTable` environment. This macro resets the list of rows for the specified table type.

`\begin{KeyValTableContent}{`⟨*tname*⟩`}`
`\end{KeyValTableContent}`

For simplifying the addition of rows, the `KeyValTableContent` 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 displaying the content collected in `KeyValTableContent` environments, 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

1.4 Setting Global Defaults

`\kvtSet{`⟨*options*⟩`}`

The keyvaltable package allows changing the default values globally for the parameters of tables and columns. This can be done by using the `\kvtSet` 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.

1.5 Row Numbering and Labeling

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=\thekvtRow;
  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.

Row numbering can easily be combined with row labeling. The following example shows how the format column property can be used for this purpose.

```
\NewKeyValTable{Labeled}{
  label: align=r, head=\textbf{\#},
         format=\kvtLabel{kvtRow};
  text: align=l, head=\textbf{Text}}
\begin{KeyValTable}{Labeled}
\Row{text=First row, label=first}
\Row{text=After row \ref{first}}
\end{KeyValTable}
```

#	Text
1	First row
2	After row 1

\kvtLabel[⟨*labelopts*⟩]{⟨*counter*⟩}{⟨*label*⟩}

The \kvtLabel macro shows the current value of the ⟨*counter*⟩ – in particular kvtRow, kvtTypeRow, and kvtTotalRow – and sets the ⟨*label*⟩ to the value of ⟨*counter*⟩. When using the macro with the format property, only the first argument (⟨*counter*⟩) must be provided, as the above example shows. The second argument (⟨*label*⟩) is provided by the respective cell content.

The \kvtLabel macro should work well with packages that change the referencing, like cleveref or varioref. When using a package that adds an optional argument to the \label command (like cleveref does), the ⟨*labelopts*⟩ can be used to pass an optional argument to \label. This feature is demonstrated in Section 2.1.

1.6 Column Spanning

Combining multiple consecutive cells in a row to a single cell (aka column spanning) can serve several purposes. The keyvaltable package supports the following purposes:

1. grouping of columns through cells in the table's header that span all cells in the group and assign a joint title to the group;
2. individual combinations of cells in the table data.

The remainder of this section describes how each of the purposes can be addressed in `KeyValTable` environments.

Column groups in table headers Column groups in table headers can be specified by the ⟨*headers*⟩ parameter of `\NewKeyValTable`. The following two examples illustrate how this parameter can be used for specifying column groups. The first example produces a single header row in which two columns are grouped with a single header, one column has a normal header, and in which one column is not provided with a header.

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

	ingredient		step
1.	150g	ice cream	put into bowl
2.	50g	cherries	heat up and add to bowl

The second example shows how multiple header rows can be specified and, particularly, how the normal column headers can be displayed through the use of "`::`".

```
\NewKeyValTable{ColGroup2}{
  date:  align=r, head=\textbf{date};
  min/Berlin: align=r, head=min;
  max/Berlin: align=r, head=max;
  min/Paris: align=r, head=min;
  max/Paris: align=r, head=max;
}[
  min/Berlin+max/Berlin+min/Paris+max/Paris:
    head=\textbf{temperature}\\
  min/Paris+max/Paris: head=\textbf{Paris};
  min/Berlin+max/Berlin: head=\textbf{Berlin}\\
  ::
]
\begin{KeyValTable}{ColGroup2}
\Row{date=01.01.1970,
    min/Berlin=0\degree C, max/Berlin=...}
\end{KeyValTable}
```

	temperature			
	Berlin		Paris	
date	min	max	min	max
01.01.1970	0°C	...		

The syntax for ⟨*headers*⟩ is as follows:

- ⟨*headers*⟩ is a list, separated by "\\", where each element in the list specifies the columns of a single header ⟨*row*⟩.
- Each ⟨*row*⟩, in turn, is also a list. The elements of this list are separated by ";" (as in the columns specification of `\NewKeyValTable`) and each element specifies a header ⟨*cell*⟩.
- Each ⟨*cell*⟩ is of the form

  ⟨*col*⟩+…+⟨*col*⟩ : ⟨*property*⟩=⟨*value*⟩ , ⟨*property*⟩=⟨*value*⟩ , …

  where each ⟨*col*⟩ is the name of a column. The specified header cell then spans each of the listed columns. The columns must be displayed consecutively, though not necessarily in the same order in which they are specified in ⟨*cell*⟩.
- The ⟨*property*⟩=⟨*value*⟩ pairs configure certain properties of the header cell. The ⟨*property*⟩ can be one of the following:

Key	Description and Possible Values	Default
`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`.	`c`
`head`	This property specifies the content of the column's header row. The default value for this property is the name of the column.	

Manual column spanning with `\multicolumn` The `\multicolumn` macro can be used for the content of a cell. The effect of this is that a number of subsequent cells are spanned over with the content of the cell. The following example demonstrates the use.

```
\NewKeyValTable{MultiCol}{
  col1: align=l;
  col2: align=l;
  col3: align=l;}
\begin{KeyValTable}{MultiCol}
  \Row{col1=1, col2=\multicolumn{1}{r}{2}, col3=3}
  \Row{col1=1, col2=\multicolumn{2}{c}{2+3}}
  \Row{col1=\multicolumn{2}{c}{1+2}, col3=3}
  \Row{col1=\multicolumn{3}{c}{1+2+3}}
\end{KeyValTable}
```

col1	col2	col3
1	2	3
1	2+3	
1+2		3
1+2+3		

A word of warning: The `\multicolumn` macro implicitly constrains the ordering of columns. For instance, in the above example, switching columns 2 and 3 would lead to an error in the second row (because `col2` is the rightmost column and therefore cannot span two columns) and also in the third row (because `col1` spans two columns but the second, `col3` is not empty). Thus, column spanning via `\multicolumn` should be used with care.

1.7 Alternative Table Environments

Originally, the keyvaltable package uses the tabu package and `tabu`, resp. `longtabu` environments for typesetting the actual tables. Through the `shape` option of tables, the table environment used by keyvaltable tables can be changed. Table 14 on the next page compares the possible shapes/environments with regards to whether they support tables that span multiple pages, whether they support `X`-type (variable-width) columns, and whether their width can be specified (through the `width` option). Finally, the table also displays the package(s) that must be loaded manually when the respective shapes are used. Examples can be found in Figure 2 on the following page.

1.8 Special Row Formatting

Through the ⟨*options*⟩ argument of the `\Row[`⟨*options*⟩`]{`⟨*content*⟩`}` and the `\KeyValRow{`⟨*tname*⟩`}[`⟨*options*⟩`]{`⟨*content*⟩`}` macros, special options of the row can be configured. As with other option arguments of the keyvaltable package, the options must be a comma-separated list of key-value pairs. The following table lists the supported option keys and their meaning.

Key	Description and Possible Values	Default
`hidden`	This property specifies whether the row shall be hidden (`true`) or not (`false`). If only `hidden` is used without a value, this is equivalent to `hidden=true`.	`false`
`bg`	This property specifies the background color for the particular row. If left empty, the default color as determined by the `rowbg` option of the table applies.	*(empty)*
`above`	This property specifies extra vertical space above the row. Note that this space is currently not colored with the row's background color but with the page's background color. The argument, if provided, is directly passed to `\vspace`.	*(empty)*
`below`	Analogously to `above`, this property specifies extra vertical space below the row.	*(empty)*
`around`	This property is a short-hand for setting both, `above` and `below`, to the same value.	*(empty)*

The following example demonstrates the options.

```
\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}
\Row[hidden]{amount=25g, ingredient=cream,
  step=decorate on top}
\Row[above=1ex,bg=red!10!white]{
  step=serve with a smile}
\end{KeyValTable}
```

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

shape	environment	multipage	X-cols	width	packages
onepage	tabu	no	yes	yes	tabu*
multipage	longtabu	yes	yes	yes	tabu*, longtable*
tabular	tabular	no	no	no	
tabularx	tabularx	no	yes	yes	tabularx
longtable	longtable	yes	no	no	longtable

Packages marked with "*" only need to be loaded if automatic loading is not disabled via the NoTabuPkg option to the keyvaltable package.

Table 14: Comparison of table shapes / environments

```
\NewKeyValTable[showrules=false]{ShapeNoX}{
  id: align=l, default=\thekvtTypeRow;
  l: align=l; c: align=c; r: align=r;}[
  l+c+r: head=\textbf{\kvtTableOpt{shape} shape}\\ ::]
\begin{KeyValTable}[shape=tabular]{ShapeNoX}
  \Row{l=left, c=center,   r=right}
  \Row{l=left-2, c=2-center-2, r=2-right}
\end{KeyValTable}\\
\begin{KeyValTable}[shape=longtable]{ShapeNoX}
  \Row{l=left, c=center,   r=right}
  \Row{l=left-2, c=2-center-2, r=2-right}
\end{KeyValTable}
```

tabular shape

id	l	c	r
1	left	center	right
2	left-2	2-center-2	2-right

longtable shape

id	l	c	r
3	left	center	right
4	left-2	2-center-2	2-right

```
\NewKeyValTable[showrules=false]{ShapeWithX}{
  id: align=l, default=\thekvtTypeRow;
  l: align=l; X: align=X; r: align=r;}[
  l+X+r: head=\textbf{\kvtTableOpt{shape} shape}\\ ::]
\begin{KeyValTable}[shape=tabularx]{ShapeWithX}
  \Row{l=left, X=expandable, r=right}
  \Row{l=left-2, X=expandable-2, r=2-right}
\end{KeyValTable}\medskip\\
\begin{KeyValTable}[shape=onepage]{ShapeWithX}
  \Row{l=left, X=expandable, r=right}
  \Row{l=left-2, X=expandable-2, r=2-right}
\end{KeyValTable}
\begin{KeyValTable}[shape=multipage]{ShapeWithX}
  \Row{l=left, X=expandable, r=right}
  \Row{l=left-2, X=expandable-2, r=2-right}
\end{KeyValTable}
```

tabularx shape

id	l	X	r
1	left	expandable	right
2	left-2	expandable-2	2-right

onepage shape

id	l	X	r
3	left	expandable	right
4	left-2	expandable-2	2-right

multipage shape

id	l	X	r
5	left	expandable	right
6	left-2	expandable-2	2-right

Figure 2: Examples for the shape option

1.9 Rules Between Rows

Additional horizontal rules between rows can simply be added by placing the respective rule command between \Row commands. The following example demonstrates this possibility.

```
\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}
\midrule
\Row{step=serve with a smile}
\end{KeyValTable}
```

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

2 Use with Other Packages

2.1 Named References

The \kvtLabel feature of the keyvaltable package can be used together with named references, as provided by the cleveref package. A name to a row label can be given by using the optional first argument to the \kvtLabel formatting macro and specifying the name to use using \crefname. The following example uses "row" for the optional argument and "line" for the displayed name of the reference.

```
\usepackage{cleveref}
\crefname{row}{line}{lines}
\NewKeyValTable[headfmt=\bfseries]{NamedRef}{
  label: align=r, head=Line,
         format=\kvtLabel[row]{kvtRow};
  text: align=l, head=Text}
\begin{KeyValTable}{NamedRef}
\Row{text=First row, label=one}
\Row{text=After \cref{one}}
\end{KeyValTable}
```

Line	Text
1	First row
2	After line 1

2.2 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 defining 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

2.3 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

3 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 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.

4 Future Work

- support for further table environments, such as xltabular: The existing code structure should make this not too complicated. Particularly for xltabular, a spurious "`missing } inserted`" error occurs.
- improved row coloring that makes sure that the alternation re-starts on continued pages of a table that spans several pages

5 Implementation

We use etoolbox for some convenience macros that make the code more easily maintainable and use xkeyval for options in key–value form. The trimspaces package is used once for trimming spaces before a string comparison.

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

We use booktabs for nice horizontal lines and xcolor for row coloring.

```
\PassOptionsToPackage{table}{xcolor}
\RequirePackage{xcolor}
\RequirePackage{booktabs}
```

5.1 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}{}}}
```

`\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@addchoicetableprop` The `\kvt@addchoicetableprop{`⟨*name*⟩`}{`⟨*default*⟩`}{`⟨*choice*⟩`}` macro adds a new table option, named ⟨*name*⟩ and with default value ⟨*default*⟩ and possible values from the comma-separated list provided by ⟨*choice*⟩.

```
\newcommand\kvt@addchoicetableprop[3]{%
  \define@choicekey[kvt]{defaults}{#1}{#3}{%
    \kvt@lazypreset{Table}{#1=##1}}%
  \presetkeys[kvt]{defaults}{#1=#2}{}%
  \define@choicekey[kvt]{Table}{#1}{#3}%
    {\csdef{cmdkvt@Table@#1}{##1}}%
  \presetkeys[kvt]{Table}{#1=#2}{}}
```

\kvt@addbooltableprop The \kvt@addbooltableprop{⟨*name*⟩}{⟨*default*⟩} macro adds a new table option, named ⟨*name*⟩ and with default value ⟨*default*⟩ and possible values being booleans (`true`, `false`).

```
\newcommand\kvt@addbooltableprop[2]{%
  \define@boolkey[kvt]{defaults}{#1}{%
    \kvt@lazypreset{Table}{#1=##1}}%
  \presetkeys[kvt]{defaults}{#1=#2}{}%
  \define@boolkey[kvt]{Table}{#1}%
    {\csdef{cmdkvt@Table@#1}{##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*⟩}{⟨*initial*⟩}{⟨*default*⟩}{⟨*choice*⟩} macro adds a new column option, named ⟨*name*⟩, with initial value ⟨*initial*⟩, with default argument value ⟨*default*⟩, and possible values from the comma-separated list provided by ⟨*choice*⟩.

```
\newcommand\kvt@addchoicecolumnprop[4]{%
```

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}{#4}[#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}{#4}[#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@addbooltableprop{showhead}{true}
```

```
\kvt@addbooltableprop{showrules}{true}
\kvt@addtableprop{headfmt}{}
\kvt@addtableprop{headalign}{}
\kvt@addtableprop{width}{\linewidth}
```

When adding further shape options below, ensure to also add a corresponding `\kvt@DefineStdTabEnv` counterpart further below in the code.

```
\kvt@addchoicetableprop{shape}{multipage}{%
  multipage,onepage,tabular,longtable,tabularx}
\kvt@addcolumnprop{default}{}
\kvt@addcolumnprop{format}{\kvtStrutted}
\kvt@addcolumnprop{align}{l}
\kvt@addcolumnprop{head}{}
\kvt@addchoicecolumnprop{hidden}{false}{true}{false,true}
\kvtSet{}
```

`\kvtTableOpt` The `\kvtTableOpt{`⟨*optname*⟩`}` macro, inside a `KeyValTable` environment, expands to the value of the table option ⟨*optname*⟩.

```
\newcommand\kvtTableOpt[1]{\csname cmdkvt@Table@#1\endcsname}
```

`\kvtStrutted` The `\kvtStrutted{`⟨*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\kvtStrutted[1]{\strut #1\ifhmode\expandafter\strut\fi}
```

5.2 Declaring Key-Value Tables

`\NewKeyValTable` The `\NewKeyValTable[`⟨*options*⟩`]{`⟨*tname*⟩`}{`⟨*colspecs*⟩`}[`⟨*headers*⟩`]` declares a new key-value table type, identified by the given ⟨*tname*⟩. The columns of the table type are specified by ⟨*colspecs*⟩. The optional ⟨*options*⟩, if given, override the default table options for tables of type ⟨*tname*⟩.

```
\newcommand\NewKeyValTable[3][]{%
  \@ifnextchar[%]
    {\kvt@NewKeyValTable{#1}{#2}{#3}}%
    {\kvt@NewKeyValTable{#1}{#2}{#3}[]}}
```

The `\kvt@NewKeyValTable{`⟨*options*⟩`}{`⟨*tname*⟩`}{`⟨*colspecs*⟩`}[`⟨*headers*⟩`]` macro is an auxiliary macro used for parsing the fourth, optional argument of `\NewKeyValTable`.

```
\def\kvt@NewKeyValTable#1#2#3[#4]{%
```

First initialize the "variables".

```
  \csdef{kvt@options@#2}{#1}%
  \csdef{kvt@headings@#2}{}%
```

The following adds a zero-width column to the left of every table. This column serves the purpose of "holding" the code that keyvaltable uses for formatting a row (e.g., parsing `\Row` arguments). This code is partly not expandable. The reason for not putting this code into the first actual colum of tables is that this code would prevent `\multicolumn` to be used in the first column. Fixme: Ideally, the whole extra column should be removed through sufficient use of `\noalign` in headers and rows, such that even the presence of `\multicolumn` does not produce errors.

```
  \csedef{kvt@alignments@#2}{p{0pt}\expandonce\kvt@HackIntercolSpace}%
  \csdef{kvt@colkeys@#2}{}%
  \csdef{kvt@rowcount@#2}{0}%
  \csdef{kvt@rows@#2}{}%
  \csdef{kvt@headings@#2}{\kvt@defaultheader}
  \listadd\kvt@alltables{#2}%
```

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{#2}##1::\@undefined}%
  \kvt@dossvlist{#3}%
```

The following terminates the argument list of `\kvt@defaultheader`.

```
  \csappto{kvt@headings@#2}{{\@nil}}%
```

Finally, parse ⟨*headers*⟩, also a semicolon-separated list of individual column groups, where `\\` marks a new row of column groups. If ⟨*headers*⟩ is omitted (or empty), then simply take the result of `\kvt@parsecolspec` as the header row.

```
  \ifstrempty{#4}
    {\csdef{kvt@headrowcount@#2}{1}}
    {\kvt@parseheadrows{#2}{#4}}%
}
```

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}}%
```

Append the column heading to `\kvt@headings@`⟨*tname*⟩, which collects arguments to `\kvt@defaultheader`. Hence, the appended tokens are enclosed in curly braces. If no `head` is specified for the column, ⟨*cname*⟩ is used for the column head. Otherwise, the `head` value is used.

```
    \ifcsvoid{kvt@col@head@#1@#2}%
      {\csappto{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@defaultheader The \kvt@defaultheader{⟨*head1*⟩}…{⟨*headn*⟩}\@nil macro, takes n header cell titles, ⟨*head1*⟩ to ⟨*headn*⟩ and formats them based on the headfmt and headalign options. More precisely, when fully expanded, \kvt@defaultheader yields "⟨*rowcolor*⟩& ⟨*fmthead1*⟩ & … & ⟨*fmtheadn*⟩\tabularnewline". In the above, ⟨*rowcolor*⟩=\rowcolor{⟨*headbg*⟩}.

```
\newcommand\kvt@defaultheader{%
  \noexpand\rowcolor{\cmdkvt@Table@headbg}%
  \kvt@defaultheader@i}
\newcommand\kvt@defaultheader@i[1]{%
  \kvt@ifnil{#1}{\noexpand\tabularnewline}{%
    \unexpanded{&}%
    \ifdefvoid\cmdkvt@Table@headalign
      {\expandonce\cmdkvt@Table@headfmt\unexpanded{#1}}
      {\noexpand\multicolumn{1}{\expandonce\cmdkvt@Table@headalign}
        {\expandonce\cmdkvt@Table@headfmt\unexpanded{#1}}}%
    \kvt@defaultheader@i}}
```

\kvt@ifnil The \kvt@ifnil{⟨*val*⟩}{⟨*iftrue*⟩}{⟨*iffalse*⟩} macro expands to ⟨*iftrue*⟩ if ⟨*val*⟩ is \@nil, and expands to ⟨*iffalse*⟩ otherwise. Fixme: The \relax in the following is not fully ideal as it is not swallowed by the \ifx and therefore remains in the macro's expansion.

```
\newcommand\kvt@ifnil[1]{%
  \ifx\@nil#1\relax
    \expandafter\@firstoftwo\else
    \expandafter\@secondoftwo\fi}
```

\kvt@HackIntercolSpace The \kvt@HackIntercolSpace macro captures the negative space that cancels out the spacing otherwise caused by the extra column that the package adds.

```
\newcommand\kvt@HackIntercolSpace{%
  @{\hspace{-.5\arrayrulewidth}}}
```

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

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

5.3 Custom Column Headers

\kvt@parseheadrows The \kvt@parseheadrows{⟨*tname*⟩}{⟨*headers*⟩} macro parses the ⟨*headers*⟩ argument of \NewKeyValTable.

```
\newcommand\kvt@parseheadrows[2]{%
  \csdef{kvt@@colgroups@#1}{}%
  \csdef{kvt@headrowcount@#1}{0}%
  \bgroup
  \def\kvt@@parseheadrows{}%
```

Now loop over ⟨*headers*⟩ to split ⟨*headers*⟩ by \\. Append each item, which specifies a single header row, to \kvt@@parseheadrows for subsequent parsing by \kvt@parseheadrow. If an item equals the special sequence ":::", then the original header for the columns is added as header row.

```
  \def\do##1{%
```

```
    \def\kvt@@tmp{##1}\trim@post@space@in\kvt@@tmp%
    \expandafter\ifstrequal\expandafter{\kvt@@tmp}{::}
      {\appto\kvt@@parseheadrows{%
         \cseappto{kvt@@colgroups@#1}{%
           \csexpandonce{kvt@headings@#1}}}}
      {\appto\kvt@@parseheadrows{\kvt@parseheadrow{#1}{##1}}}%
```

Increment the header row counter for each \\-separated item of ⟨*headers*⟩.

```
    \appto\kvt@@parseheadrows{\csedef{kvt@headrowcount@#1}{%
      \the\numexpr\csuse{kvt@headrowcount@#1}+1\relax}}%
  }\kvt@dobrklist{#2}%
```

Finally, escape the inner group and overwrite the headings with the result of the parsing.

```
  \expandafter\egroup\kvt@@parseheadrows
  \csletcs{kvt@headings@#1}{kvt@@colgroups@#1}}
```

\kvt@parseheadrow The \kvt@parseheadrow{⟨*tname*⟩}{⟨*colspec*⟩} macro parses a single header row and appends the resulting table code to \kvt@@colgroups@⟨*tname*⟩.

```
\newcommand\kvt@parseheadrow[2]{%
  \bgroup
```

First parse ⟨*colspec*⟩, populating the \kvt@@colgrpof@⟨*colname*⟩ macros that associate each column with the column group to which the column belongs.

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

Initialize variables for the subsequent loop. The \kvt@@tmpgrphd macro collects the code for the cells of the current header row. The \kvt@@span counter specifies how many columns the current cell shall span. Finally, \kvt@@curgrp and \kvt@@lastgrp hold the name of the group in which the current column and, respectively, previous column are in. Each of the two macros is undefined if there is no such column group.

```
  \let\kvt@@tmpgrphd\@empty
  \kvt@@span\z@
  \undef\kvt@@curgrp \undef\kvt@@lastgrp
```

Next, loop over all displayed (non-hidden) columns stored in \kvt@colkeys@⟨*tname*⟩. The following \do{⟨*colname*⟩} collects (spanned) columns as specified in ⟨*colspec*⟩, in the ordering in which the table's columns are displayed. The spanned columns are stored in \kvt@@tmpgrphd.

```
  \def\do##1{\letcs\kvt@@curgrp{kvt@@colgrpof@##1}%
    \ifdefequal\kvt@@curgrp\kvt@@lastgrp
```

If the column group has not changed, simply increase the spanning counter.

```
      {\advance\kvt@@span\@ne}%
```

Otherwise, i.e., if the column group has changed, then conclude the previous column (if there was one) and reset the span to 1 (to count for the column in \kvt@@curgrp) and set \kvt@@lastgrp to the current one.

```
      {\ifnum\kvt@@span>\z@ \expandafter\kvt@concludecolumn\fi
       \ifdefvoid\kvt@@curgrp{}{\ifcsdef{kvt@@colgrpdone@\kvt@@curgrp}{%
         \kvt@error{Column group `\kvt@@curgrp' must consist of only
```

```
            consecutive columns, but it is not}%
          {Compare `|\kvt@@curgrp|' to the column ordering as specified
           in `\string\NewKeyValTable{#1}'}}{}}%
      \kvt@@span\@ne \let\kvt@@lastgrp\kvt@@curgrp}%
  }\dolistcsloop{kvt@colkeys@#1}%
  \kvt@concludecolumn
```

Finally, conclude the whole header row and append the row to the overall list of rows, stored in `\kvt@@colgroups@`⟨*tname*⟩, while ending the current TEX group.

```
  \appto\kvt@@tmpgrphd{\tabularnewline}%
  \edef\do{\noexpand\csappto{kvt@@colgroups@#1}{%
    \noexpand\noexpand\noexpand\rowcolor{\noexpand\cmdkvt@Table@headbg}%
    \noexpand\unexpanded{\expandonce{\kvt@@tmpgrphd}}}}%
  \expandafter\egroup\do}
```

`\kvt@@span` The counter `\kvt@@span` is used temporarily in macros for counting how many columns are spanned by column groups.

```
\newcount\kvt@@span
```

`\kvt@concludecolumn` The `\kvt@concludecolumn` macro appends a cell, potentially spanning multiple columns, to the row under construction (which is in `\kvt@@tmpgrphd`).

```
\newcommand\kvt@concludecolumn{%
```

The following conditional checks whether this is the first column group in the header row. If this is the case, then the `\kvt@@extraalign` macro is set to `\kvt@HackIntercolSpace`, such that the `\multicolumn` below does not throw away this spacing.

```
  \ifdefequal\kvt@@tmpgrphd\@empty
    {\let\kvt@@extraalign\kvt@HackIntercolSpace}
    {\let\kvt@@extraalign\@empty}%
  \appto\kvt@@tmpgrphd{&}%
  \ifdefvoid\kvt@@lastgrp{}{%
    \eappto\kvt@@tmpgrphd{\noexpand\multicolumn
      {\the\kvt@@span}
      {\expandonce\kvt@@extraalign
       \csexpandonce{kvt@@colgrp@align@\kvt@@lastgrp}}
      {\csexpandonce{kvt@@colgrp@head@\kvt@@lastgrp}}}%
```

Mark the column group as already used and concluded, such that another use of the same column group can be detected and raise an error.

```
    \cslet{kvt@@colgrpdone@\kvt@@lastgrp}{\@ne}}}
```

`\kvt@parsehdcolspec` The `\kvt@parsehdcolspec{`⟨*tname*⟩`}`⟨*cname*⟩`:`⟨*config*⟩`:`⟨*empty*⟩`\@undefined` macro parses a single header column (resp. column group), ⟨*cname*⟩. For a column group, ⟨*cname*⟩ can consist of multiple, "+"-separated column names.

```
\def\kvt@parsehdcolspec#1#2:#3:#4\@undefined{%
```

First link the individual columns of a column group to the group. In this, ensure that no column is contained in more than one column group.

```
  \def\kvt@@colreg##1{%
    \ifinlistcs{##1}{kvt@colkeys@#1}{}
      {\kvt@error{Column `##1' referenced in column group `#2' not known
```

```
        in table type `#1'}{Check the \string\NewKeyValTable{#1} for
        the names of known columns and check `##1' for a typo.}}%
    \ifcsmacro{kvt@@colgrpof@##1}
      {\kvt@error{Column `##1' used in more than one column group}
         {Check the fourth, optional argument of \string\NewKeyValTable
         and eliminate multiple occurrences of column `##1'.}}
      {\csdef{kvt@@colgrpof@##1}{#2}}%
  }\kvt@forpsvlist{\kvt@@colreg}{#2}%
```

Now parse the ⟨*config*⟩ of the column, resp. column group.

```
  \def\kvt@@colgrp{#2}%
  \setkeys[kvt]{ColGroup}{#3}}
```

The following defines the options for header cells.

```
\define@key[kvt]{ColGroup}{head}{%
  \csdef{kvt@@colgrp@head@\kvt@@colgrp}{#1}}
\define@key[kvt]{ColGroup}{align}{%
  \csdef{kvt@@colgrp@align@\kvt@@colgrp}{#1}}
\presetkeys[kvt]{ColGroup}{align=c}{}%
```

5.4 Row Numbering and Labeling

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}{0}
```

`\kvtLabel` The `\kvtLabel[`⟨*labelopts*⟩`]{`⟨*counter*⟩`}{`⟨*label*⟩`}` macro sets a label, named ⟨*label*⟩, for the current value of the LaTeX counter named ⟨*counter*⟩.

```
\newcommand\kvtLabel[3][]{%
```

The following imitates a `\refstepcounter` in the sense of setting the current label, but it does not touch the ⟨*counter*⟩ (in case someone added some custom hooks to them).

```
  \setcounter{kvt@LabelCtr}{\value{#2}}%
```

```
197   \addtocounter{kvt@LabelCtr}{-1}%
198   \refstepcounter{kvt@LabelCtr}%
```

Next, define the ⟨*label*⟩ (if provided) and show the value of ⟨*counter*⟩.

```
199   \ifstrempty{#3}{}{%
200     \ifstrempty{#1}{\label{#3}}{\label[#1]{#3}}}%
201   \csuse{the#2}}
```

kvt@LabelCtr The `kvt@LabelCtr` counter is an auxiliary counter for setting labels, used by `\kvtLabel`.

```
202 \newcounter{kvt@LabelCtr}
```

5.5 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*⟩.

```
203 \newenvironment{KeyValTable}[2][]{%
204   \bgroup%
```

\Row The `\Row[`⟨*options*⟩`]{`⟨*content*⟩`}` macro is made available locally in the `KeyValTable` environment.

```
205   \def\Row{\kvt@AddKeyValRow
206     {\noalign\bgroup}{\expandafter\egroup\kvt@@row}{#2}}%

207   \kvt@SetOptions{#2}{#1}%
208   \csuse{kvt@StartTable@\cmdkvt@Table@shape}{#2}%
209 }{%
210   \csuse{kvt@EndTable@\cmdkvt@Table@shape}%
211   \egroup}
```

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

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

\kvt@SetOptions The `\kvt@SetOptions{`⟨*tname*⟩`}{`⟨*options*⟩`}` set the specific table options in the current environment, based on the options for table type ⟨*tname*⟩ and the specific ⟨*options*⟩.

```
214 \newcommand\kvt@SetOptions[2]{%
215   \bgroup\edef\kvt@@do{\egroup\noexpand%
216     \setkeys[kvt]{Table}%
217       {\csexpandonce{kvt@options@#1},\unexpanded{#2}}%
218   }\kvt@@do}
```

5.5.1 Table Environment Code

\kvt@StartTabularlike The `\kvt@StartTabularlike{`⟨*env*⟩`}{`⟨*tname*⟩`}{`⟨*bLong*⟩`}{`⟨*bTabu*⟩`}{`⟨*bWidth*⟩`}` macro begins a table environment for the given table type ⟨*tname*⟩. The ⟨*env*⟩ parameter specifies the concrete environment name. The parameters ⟨*bLong*⟩, ⟨*bTabu*⟩, and

⟨*bWidth*⟩ are Boolean parameters (expecting value `true` or value `false`). They specify whether the table environment supports multi-page tables (⟨*bLong*⟩), whether the environment is a tabu environment (⟨*bTabu*⟩), and whether the environment supports specifying the width of the table (⟨*bWidth*⟩).

```
\newcommand\kvt@StartTabularlike[5]{%
```

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

```
  \gdef\kvt@@recenttable{#2}%
  \ifbool{kvt@Table@showrules}
    {\def\kvt@@rule##1{\csuse{##1rule}}}
    {\def\kvt@@rule##1{}}%
  \csuse{kvt@@patchenvend@#1}%
  \setcounter{kvtRow}{0}%
  \setcounter{kvtTypeRow}{\csuse{kvt@rowcount@#2}}%
```

In `\kvt@@do`, the start code for the environment, including the header rows, is gathered, with expansion to fill in all the table settings and options.

```
  \bgroup\edef\kvt@@do{\egroup
    \ifbool{#4}{}{\noexpand\kvt@dottedrowcolors
      {\ifbool{kvt@Table@showhead}
        {\the\numexpr\csuse{kvt@headrowcount@#2}+1\relax}
        {1}}%
      {\expandonce\cmdkvt@Table@rowbg}}%
    \expandafter\noexpand\csname #1\endcsname
      \ifbool{#5}
        {\ifbool{#4}
          {to \expandonce\cmdkvt@Table@width}
          {{\expandonce\cmdkvt@Table@width}}}
        {}%
      {\csexpandonce{kvt@alignments@#2}}%
    \noexpand\kvt@@rule{top}%
    \ifbool{kvt@Table@showhead}
      {\csuse{kvt@headings@#2}\noexpand\kvt@@rule{mid}}
      {}%
    \ifbool{#4}
      {\noexpand\taburowcolors 2{\expandonce\cmdkvt@Table@rowbg}}{}%
    \ifbool{#3}{\noexpand\endhead}{}%
  }\kvt@@do}
```

`\kvt@stepcounters` The `\kvt@stepcounters[`⟨*delta*⟩`]` macro increments all row counters by ⟨*delta*⟩. If ⟨*delta*⟩ is omitted, ⟨*delta*⟩=1.

```
\newcommand\kvt@stepcounters[1][1]{%
  \addtocounter{kvtRow}{#1}%
  \addtocounter{kvtTypeRow}{#1}%
  \addtocounter{kvtTotalRow}{#1}}
```

`\kvt@DefineStdTabEnv` The `\kvt@DefineStdTabEnv[`⟨*shape*⟩`]{`⟨*env*⟩`}{`⟨*bLong*⟩`}{`⟨*bTabu*⟩`}{`⟨*bWidth*⟩`}{`⟨*endpatch*⟩`}` macro defines the macros needed for the given ⟨*shape*⟩ value. If ⟨*shape*⟩ is omitted,

⟨*env*⟩ (the name of the environment to use for the shape) is used as ⟨*shape*⟩ value. The ⟨*endpatch*⟩ parameter expects macro code that shall be run at the beginning of the `KeyValTable` environment to (locally) patch macros related to the end code of ⟨*env*⟩ for ensuring that the bottom rule, `\kvt@@rule{bottom}`, is displayed. If ⟨*endpatch*⟩ is empty, the rule is displayed via `\kvt@EndTable@`⟨*shape*⟩. Otherwise, `\kvt@EndTable@`⟨*shape*⟩ equals `\end`⟨*env*⟩. Environments such as `tabularx` require the latter to parse for the end of the environment. The parameters ⟨*bLong*⟩, ⟨*bTabu*⟩, and ⟨*bWidth*⟩ are the same as for `\kvt@StartTabularlike`.

Note: In the future, the macro could automatically add ⟨*option*⟩ to the list of possible values for the `shape` option.

```
\newcommand\kvt@DefineStdTabEnv{\@dblarg\kvt@DefineStdTabEnv@i}
\newcommand\kvt@DefineStdTabEnv@i[6][]{%
  \expandafter\newcommand\csname kvt@StartTable@#1\endcsname[1]{%
    \kvt@StartTabularlike{#2}{##1}{#3}{#4}{#5}}%
  \csedef{kvt@EndTable@#1}{%
    \ifstrempty{#6}{\noexpand\kvt@@rule{bottom}}{}%
    \expandafter\noexpand\csname end#2\endcsname}%
  \ifstrempty{#6}{}{\csdef{kvt@@patchenvend@#2}{#6}}}
```

The following lines define the macros for the various table shapes / environments.

```
\kvt@DefineStdTabEnv{tabular}{false}{false}{false}{}
\kvt@DefineStdTabEnv{longtable}{true}{false}{false}{}
\kvt@DefineStdTabEnv{tabularx}{false}{false}{true}{%
  \preto\TX@endtabularx{\toks@\expandafter{\the\toks@
      \kvt@@rule{bottom}}}}
\kvt@DefineStdTabEnv{xltabular}{true}{false}{true}{%
  \preto\XLT@ii@TX@endtabularx{\toks@\expandafter{\the\toks@
      \kvt@@rule{bottom}}}}
\kvt@DefineStdTabEnv[onepage]{tabu}{false}{true}{true}{}
\kvt@DefineStdTabEnv[multipage]{longtabu}{true}{true}{true}{}
```

`\kvt@dottedrowcolors` The `\kvt@dottedrowcolors{`⟨*start-row*⟩`}{`⟨*colors*⟩`}` sets up row colors using the `\rowcolors` macro of xcolor. The `{`⟨*colors*⟩`}` parameter expects arguments of the form "⟨*color1*⟩`..`⟨*color2*⟩" (the syntax used for the `rowbg` option. The row colors then alternate between ⟨*color1*⟩ and ⟨*color2*⟩, starting with ⟨*color1*⟩ in ⟨*start-row*⟩. This macro substitutes `\taburowcolors` for non-tabu environments.

```
\newcommand\kvt@dottedrowcolors[2]{%
  \kvt@dottedrowcolors@i{#1}#2\@nil}
\def\kvt@dottedrowcolors@i#1#2..#3\@nil{%
```

Since `\rowcolors` expects its color arguments to specify the odd and even color, we swap arguments depending on the parity of ⟨*start-row*⟩ to ensure ⟨*color1*⟩ is applied to ⟨*start-row*⟩.

```
  \ifnumodd{#1}
    {\rowcolors{#1}{#2}{#3}}
    {\rowcolors{#1}{#3}{#2}}}
```

5.5.2 Environment-Independent Parts

The following block declares the known row options.

```
\define@cmdkey[kvt]{Row}{bg}{}%
\define@boolkey[kvt]{Row}{hidden}[true]{}%
\define@cmdkey[kvt]{Row}{below}{}
\define@cmdkey[kvt]{Row}{above}{}
\define@cmdkey[kvt]{Row}{around}{%
  \def\cmdkvt@Row@above{#1}\def\cmdkvt@Row@below{#1}}
```

`\kvt@AddKeyValRow` The `\kvt@AddKeyValRow{`⟨*pre*⟩`}{`⟨*post*⟩`}{`⟨*tname*⟩`}[`⟨*options*⟩`]{`⟨*content*⟩`}` macro composes a row for the table of type ⟨*tname*⟩ from the given ⟨*content*⟩ and ⟨*options*⟩. The ⟨*content*⟩ is a key-value list that specifies the content of the individual cells in the row. The result is returned in macro `\kvt@@row`. The arguments ⟨*pre*⟩ and ⟨*post*⟩ are expanded at the very beginning, resp. end of the macro. They allow to control grouping (`\bgroup` and `\egroup`) as well as table placement via `\noalign`.

```
\newcommand\kvt@AddKeyValRow[3]{%
  #1%
```

It's essential that ⟨*pre*⟩ above comes even before `\@ifnextchar` and, therefore, cannot be moved into `\kvt@AddKeyValRow@i`: The `\@ifnextchar` is not fully expandable and therefore any `\noalign` (in ⟨*pre*⟩) following `\@ifnextchar` would lead to "misplaced `\noalign`" errors.

```
  \@ifnextchar[%]
    {\kvt@AddKeyValRow@i{#2}{#3}}
    {\kvt@AddKeyValRow@i{#2}{#3}[]}}
```

`\kvt@AddKeyValRow@i` The `\kvt@AddKeyValRow@i{`⟨*post*⟩`}{`⟨*tname*⟩`}[`⟨*options*⟩`]{`⟨*content*⟩`}` macro parses ⟨*options*⟩ and evaluates the `hidden` option.

```
\def\kvt@AddKeyValRow@i#1#2[#3]#4{%
  \setkeys[kvt]{Row}{#3}%
  \ifbool{kvt@Row@hidden}
    {\let\kvt@@row\@empty #1}
    {\kvt@AddKeyValRow@ii{#1}{#2}{#4}}}
```

`\kvt@AddKeyValRow@ii` The `\kvt@AddKeyValRow@ii{`⟨*post*⟩`}{`⟨*tname*⟩`}{`⟨*content*⟩`}` macro mainly processes ⟨*content*⟩ as well as ⟨*options*⟩ that have already been parsed by `\kvt@AddKeyValRow@i`.

```
\def\kvt@AddKeyValRow@ii#1#2#3{%
  \setkeys[KeyValTable]{#2}{#3}%
```

Initialize and first add the `\noalign` material to the row.

```
  \def\kvt@@row{}%
  \ifdefvoid\cmdkvt@Row@above{}{%
    \eappto\kvt@@row{\noexpand\noalign{\noexpand\vspace{%
      \expandonce\cmdkvt@Row@above}}}%
  \ifdefvoid\cmdkvt@Row@bg{}{%
    \eappto\kvt@@row{\noexpand\rowcolor{\expandonce\cmdkvt@Row@bg}}}%
```

Place the `everyrow` hook after `\noalign`.

```
  \expandafter\appto\expandafter\kvt@@row\expandafter{\kvt@@everyrow}
```

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 that currently the default value is formatted using the given format macro – a design decision.

```
\kvt@@span=0\relax
\def\do##1{%
```

First recover the cell content (either the specified value for the row or, if no value is specified for the row, the cell's default value) without formatting.

```
  \ifcsvoid{cmdKeyValTable@#2@##1}
    {\letcs\kvt@@cell{kvt@col@default@#2@##1}}
    {\letcs\kvt@@cell{cmdKeyValTable@#2@##1}}%
```

Separately also already create the formatted content.

```
  \edef\kvt@@fmtcell{\csexpandonce{kvt@col@format@#2@##1}{%
    \expandonce\kvt@@cell}}%
```

Next, check whether a column-spanning cell is active (\kvt@@span > 0). If this is the case, ensure that if the raw cell content in the current column is empty, then formatting does not make the cell non-empty and, thereby, cause errors with the active column-spanning cell.

```
  \ifnumgreater\kvt@@span{0}
    {\advance\kvt@@span\m@ne
     \ifstrempty\kvt@@cell{\def\kvt@@fmtcell{}}{}}
    {\appto\kvt@@row{&}}%
```

Now check whether the cell itself spans multiple columns.

```
  \expandafter\kvt@CheckMulticolumn\kvt@@cell
    \relax\relax\relax\relax\@undefined{#2}{##1}%
  \expandafter\appto\expandafter\kvt@@row\expandafter{\kvt@@fmtcell}%
}\dolistcsloop{kvt@colkeys@#2}%
```

Finally, add the concluding newline for the row as well as the vertical space after the row, if requested.

```
\appto\kvt@@row{\tabularnewline}%
\ifdefvoid\cmdkvt@Row@below{}{%
  \eappto\kvt@@row{\noexpand\noalign{\noexpand\vspace{%
    \expandonce\cmdkvt@Row@below}}}%
```

At the very end of the expansion text, put ⟨*post*⟩.

```
#1}
```

\kvt@everyrow The \kvt@everyrow{⟨*code*⟩} registers ⟨*code*⟩ to be included in the first (invisible) cell of every row.

```
\newcommand\kvt@everyrow[1]{\def\kvt@@everyrow{#1}}
\newcommand\kvt@@everyrow{}
```

Initialize the hook for every row to increment the row counters.

```
\kvt@everyrow{\kvt@stepcounters}%
```

\kvt@CheckMulticolumn The \kvt@CheckMulticolumn{⟨*arg1*⟩}{⟨*arg2*⟩}{⟨*arg3*⟩}{⟨*arg4*⟩}\@undefined {⟨*tname*⟩}{⟨*colname*⟩} macro checks whether a cell's initial content (captured by ⟨*arg1*⟩ to ⟨*arg4*⟩, starts a multi-column cell. If this is the case, the macro records the arguments to \multicolumn for use by \kvt@AddKeyValRow. In this case,

⟨*arg1*⟩=\multicolumn, ⟨*arg2*⟩=⟨*n*⟩ (number of columns to span), ⟨*arg3*⟩=⟨*format*⟩ (column alignment), and ⟨*arg4*⟩=⟨*item*⟩ (the content of the cell).

```
\def\kvt@CheckMulticolumn#1#2#3#4\@undefined#5#6{%
  \ifx#1\multicolumn
```

First, record ⟨*n*⟩ in \kvt@@span. The subtraction of −1 is already in preparation for the next column, in which one spanning has already been reduced.

```
    \kvt@@span=#2\relax \advance\kvt@@span\m@ne
```

Second, move the defined cell format to inside of the ⟨*item*⟩ argument of \multicolumn rather than around the \multicolumn.

```
    \edef\kvt@@fmtcell{\unexpanded{\multicolumn{#2}{#3}}%
      {\csexpandonce{kvt@col@format@#5@#6}{\expandonce{#4}}}}%
  \fi}
```

5.6 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][]{%
  \begin{KeyValTable}[#1]{#2}%
    \csuse{kvt@rows@#2}%
  \end{KeyValTable}%
  \csdef{kvt@rows@#2}{}}
```

\AddKeyValRow The \AddKeyValRow{⟨*tname*⟩}[⟨*options*⟩]{⟨*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*⟩ and ⟨*options*⟩ parameters are the same as with \kvt@AddKeyValRow. The resulting row (\kvt@@row) is globally appended to \kvt@rows@⟨*tname*⟩.

```
\newcommand\AddKeyValRow[1]{%
  \kvt@AddKeyValRow
    {\bgroup}
    {\csxappto{kvt@rows@#1}{\expandonce{\kvt@@row}}\egroup}
    {#1}}
```

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*⟩.

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

5.7 Package Options

The tabu is used by default for typesetting the tables, additionally with longtable for tables that can span multiple pages. If the default packages are never used or the tabu package shall be loaded manually, the noTabuPkg option can be used.

```
\define@boolkey[kvt]{PackageOptions}[kvt@@]{noTabuPkg}[true]{}
```

Next, set default package options and process them.

```
\ExecuteOptionsX[kvt]<PackageOptions>{%
  noTabuPkg=false,
}
\ProcessOptionsX[kvt]<PackageOptions>\relax
```

Finally, implement the outcome of the options parsing.

```
\ifbool{kvt@@noTabuPkg}{}{%
  \RequirePackage{longtable,tabu}}
```

5.8 Auxiliary Code

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

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

\kvt@forpsvlist The \kvt@forpsvlist{⟨*handler*⟩}{⟨*list*⟩} parses a '+'-separated list.

```
\DeclareListParser*{\kvt@forpsvlist}{+}
```

\kvt@dobrklist The \kvt@dobrklist{⟨*list*⟩} parses a '\\'-separated list.

```
\DeclareListParser{\kvt@dobrklist}{\\}
```

\kvt@error
\kvt@warn

```
\newcommand\kvt@error[2]{\PackageError{keyvaltable}{#1}{#2}}
\newcommand\kvt@warn[1]{\PackageWarning{keyvaltable}{#1}}
```

Change History

Index

K

L

M

N

P

R

S

T

U

V

X

Z