The keyvaltable package*

Richard Grewe
r-g+tex@posteo.net

May 31, 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

1 Basic Usage 2
2 Defining Table Types 2
3 Typesetting Tables 3
4 Row Numbering & Labeling 6
5 Changing the Appearance 8
6 Customizing the Layout 13
7 Use with Other Packages 17
8 Related Packages 21
9 Future Work 21
10 Implementation 22

*This document corresponds to keyvaltable v2.0, dated 2019/05/11. The package is available online at http://www.ctan.org/pkg/keyvaltable and https://github.com/Ri-Ga/keyvaltable.

1 Basic 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;
}
\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` alignment is provided by the tabularx package (see the documentation there).

After defining 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. In the parameter of the `\Row` macro, the content of the individual cells can be specified by key-value 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 documentation.

To quickly sketch a table type, one can even omit properties of columns and just list their names, separated by semicolons, as the following example shows. All columns then get the default alignment: `l`.

```
\NewKeyValTable{Recipe}{amount;ingredient;step}
\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

2 Defining Table Types

As the example in Section 1 shows, `\NewKeyValTable` defines a table type.

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

The macro defines a table type with name ⟨*tname*⟩ whose columns are specified by ⟨*colspecs*⟩. The ⟨*colspecs*⟩ parameter must be a semicolon-separated list. 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` = `l`, `c`, `r`, `p`, `X`, ... *initially:* `l`

This property specifies the alignment of content in the column. The ⟨*value*⟩ can be set to any column alignment understood by table environments.

`default` = ⟨*content*⟩ *initially:* ⟨*empty*⟩

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

`format` = ⟨*single argument macro*⟩ *initially:* `\kvtStrutted`

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. Initially, the format is defined to take the content as is but puts a `\strut` before and after the content (to yield a better vertical row spacing).

`head` = ⟨*content*⟩ *initially:* ⟨*colname*⟩

This property specifies the ⟨*content*⟩ of the column's header row. The initial value for this property is the name of the column.

`hidden` = `true`, `false` *default:* `true`, *initially:* `false`

This property specifies whether a table column shall be displayed or not. The ⟨*value*⟩ for this property can be `true` (to hide the cell) or `false` (to display the cell). Using `hidden` without ⟨*value*⟩ is equivalent to specifying `hidden=true`.

The following example shows all of the above column properties in action.

```
\NewKeyValTable{ShoppingList}{
  what:   head=article, format=\textbf;
  amount: align=r, default=1;
  why:    hidden;
}
\begin{KeyValTable}{ShoppingList}
\Row{what=melon}
\Row{what=apples, amount=6}
\Row{what=bicycle, why=Bob's birthday}
\end{KeyValTable}
```

article	amount
melon	1
apples	6
bicycle	1

The ⟨*options*⟩ and ⟨*layout*⟩ parameters of `\NewKeyValTable` are described in Section 5.1 and, respectively, Section 6.1 of this documentation.

3 Typesetting Tables

The keyvaltable package offers three possibilities for typesetting tables. The first is in the traditional LaTeX form, in which there is an environment that encloses the individual row specifications. The second possibility is to specify rows throughout the document, bind them to a name, and finally typeset a table from all rows bound to the particular name. The third possibility is to source the row specifications from a file.

3.1 Specifying Rows in a Table Environment

The first possibility for typesetting a table using the keyvaltable package, is via the KeyValTable environment. Section 1 presents an example of this possibility.

\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*⟩ are described in Section 5.1.

\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 5.3.

3.2 Tables of Collected Rows

The content of a table's rows might logically belong to locations that are scattered throughout a document, e.g., to individual sections of the document. In this situation, it can be convenient to have the rows specified close to the locations their contents belong to, instead of specified in the table environment.

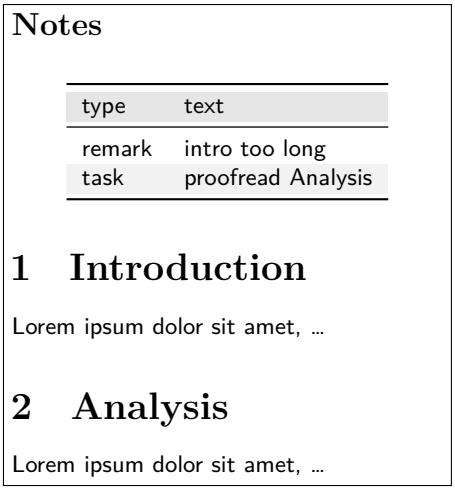
The following example illustrates the use of this feature for taking and collecting notes in a document:

```
\NewKeyValTable{Notes}{type; text}
\NewCollectedTable{notes}{Notes}

\subsection*{Notes}
\ShowCollectedTable{notes}

\section{Introduction}
\CollectRow{notes}{type=remark, text=intro too long}
Lorem ipsum dolor sit amet, \ldots

\section{Analysis}
\CollectRow{notes}{type=task, text=proofread Analysis}
Lorem ipsum dolor sit amet, \ldots
```



See Section 4.3 on how to (automatically) include references to, e.g., section or page numbers in tables. The key macros (highlighted in bold font) used in the example are the following three.

\NewCollectedTable{⟨*cname*⟩}{⟨*tname*⟩}

This macro defines the name ⟨*cname*⟩ for a new collection of rows. The collection is associated with the table type ⟨*tname*⟩. This macro must be used before `\CollectRow` for a ⟨*cname*⟩.

`\CollectRow[`⟨*options*⟩`]{`⟨*cname*⟩`}{`⟨*content*⟩`}`

This macro adds the row content ⟨*content*⟩ and row options ⟨*options*⟩ to the row collection ⟨*cname*⟩.

`\ShowCollectedTable[`⟨*options*⟩`]{`⟨*cname*⟩`}`

This macro typesets a table of the row collection ⟨*cname*⟩, with the table options ⟨*options*⟩. The table includes rows that are collected only afterwards in the document. For this, LaTeX must be run at least two times.

3.3 Sourcing Rows From a File

Rather than specifying the rows of a table inside a `KeyValTable` environment, the rows can also be sourced from a file. More concretely, this file must consist of the `\Row` macros that specify the content of the rows. For information on how to source rows from CSV files, see Section 7.2.

`\ShowKeyValTableFile[`⟨*options*⟩`]{`⟨*tname*⟩`}{`⟨*filename*⟩`}`

This macro produces a `KeyValTable` environment of type ⟨*tname*⟩ whose content is taken from the file ⟨*filename*⟩. The ⟨*options*⟩ specify the table options, which are directly passed to the options argument of the `KeyValTable` environment.

```
\begin{filecontents}{snowman.kvt}
\Row{amount=3, ingredient=balls of snow,
    step=staple all 3 balls}
\Row{amount=1, ingredient=carrot,
    step=stick into top ball}
\Row{amount=2, ingredient=coffee beans,
    step=put diagonally above carrot}
\end{filecontents}
\ShowKeyValTableFile{Recipe}{snowman.kvt}
```

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

3.4 Tables of Collected Rows (Legacy Interface)

This section documents legacy functionality of keyvaltable, that is now superseded by the functionality described in Section 3.2. The legacy functionality compares to the new functionality as follows:

- Rows must be collected *before* the place in the document where they are displayed in a table.
- For each table type, there can be only one collection of rows. After the collection has been typeset in a table the collection is emptied again.
- Row content is not written into the aux file. This might be relevant for very large tables.

The following macros and environments implement the functionality.

`\AddKeyValRow{`⟨*tname*⟩`}[`⟨*options*⟩`]{`⟨*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 3.

\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

4 Row Numbering & Labeling

The mechanism of default column values enables a simple means for automatic row numbering, labeling, and referencing document entities.

4.1 Row Numbering

For row numbering, 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[headformat=\textbf]{Numbered}{
  line: align=r, head=\#,
       format=\kvtStrutted[\textbf],
       default=\thekvtRow;
  text: align=l, head=Text}
\begin{KeyValTable}{Numbered}
\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.

By default, all rows are counted by the aforementioned counters. However, this default can be changed.

`uncounted` = `true`, `false` *default:* `true`, *initially:* `false`

This row option specifies whether the row shall not be counted (`true`) or shall be counted (`false`). If only `uncounted` is used without a value, this is equivalent to `uncounted=true`. The following example illustrates the option.

```
\begin{KeyValTable}{Numbered}
\Row{text=First row}
\Row[uncounted]{line={--}, text=interlude}
\Row{text=Second row}
\end{KeyValTable}
```

#	Text
1	First row
–	interlude
2	Second row

4.2 Row Labeling

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 7.1.

4.3 Referencing in Collected Rows

The example in Section 3.2 illustrates well a situation in which referencing the locations in the document at which rows are collected. The following example augments the original example to achieve exactly this.

```
\NewKeyValTable{Notes2}{
  id: default=\thekvtRow.;
  type; text;
  where: default={\S\thesection\ (p.\@\thepage)};}
\NewCollectedTable{notes2}{Notes2}

\subsection*{Notes}
\ShowCollectedTable{notes2}

\section{Introduction}
\CollectRow{notes2}{type=remark, text=intro too long}
Lorem ipsum dolor sit amet, \ldots

\section{Analysis}
\CollectRow{notes2}{type=task, text=proofread!}
Lorem ipsum dolor sit amet, \ldots
```

Notes

id	type	text	where
1.	remark	intro too long	§1 (p.8)
2.	task	proofread!	§2 (p.8)

1 Introduction

Lorem ipsum dolor sit amet, …

2 Analysis

Lorem ipsum dolor sit amet, …

The keyvaltable package is carefully designed to take the values of counters such as the page counter and the section counter from the point in the document where `\CollectRow` is used. At the same time, the table row counters are taken from the point inside the respective table. This applies to `\thekvtRow` as well as to `\arabic{kvtRow}` and other counter formats. For customizing this behavior, the following three macros can be used.

`\kvtDeclareTableMacros{`⟨*macro-list*⟩`}`
`\kvtDeclareTableCounters{`⟨*counter-list*⟩`}`

These macros take a comma-separated list of macros (respectively counters) and declares these as "table macros" ("table counters"). A macro or counter declared this way is expanded only inside the table environment and not at the point where `\CollectRow` is used. The keyvaltable already declares `\thekvtRow`, `\thekvtTypeRow`, and `\thekvtTotalRow` as table macros and declares `kvtRow`, `kvtTypeRow`, and `kvtTotalRow` as table counters.

`\kvtDeclareCtrFormatters{`⟨*macro-list*⟩`}`

This macro takes a comma-separated list of macros and declares them as macros for formatting counter values. Examples for such macros are `\arabic`, `\alph`, `\Alph`, `\roman`, `\Roman`, `\fnsymbol`, which keyvaltable already declares. When other counter-formatting macros shall be used in the `default` value of a column, such as `\ordinal` of the fmtcount package, they have to be passed to `\kvtDeclareCtrFormatters` first.

5 Changing the Appearance

The appearance (e.g., colors, rules) of a table can be changed at the level of the overall table as well as for individual rows, columns, and cells.

5.1 Table Appearance

The appearance of a table can be configured through the ⟨*options*⟩ parameters of

- `KeyValTable`, `\ShowKeyValTable`, and `\ShowKeyValTableFile` (affecting the particular table),
- `\NewKeyValTable` (affecting all tables of the table type), and
- `\kvtSet` (affecting all tables).

In this list, the former take precedence over the latter. That is, table options override table type options and table type options override global options for all tables.

In each case, ⟨*options*⟩ must be specified as a comma-separated list of ⟨*property*⟩=⟨*value*⟩ pairs. The following ⟨*property*⟩ keys can be configured.

`shape` = `multipage`, `onepage`, `tabular`, `tabularx`, `longtable`, `xltabular`, `tabu`, `longtabu` — *initially:* `multipage`

This property specifies the table's shape. For ⟨*value*⟩, the package currently supports `multipage` and `onepage` as well as `tabular`, `tabularx`, `longtable`, `xltabular`, `tabu`, and `longtabu`. 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 values use the respective environment for producing the table (see Section 6.4 for the effect).

`width` = ⟨*dimension*⟩ — *initially:* `\linewidth`

This property specifies the width of the table, if the selected `shape` supports it (see Section 6.4).

`showhead` = `true`, `false` — *initially:* `true`

This property specifies whether the header row shall be shown. The ⟨*value*⟩ must be a Boolean (i.e., `true` or `false`), where `true` specifies that the header row is shown and `false` specifies that the header row is not shown.

`showrules` = `true`, `false` — *initially:* `true`

This property specifies whether top and bottom rules as well as a rule below the header row are drawn (`true`) or not (`false`).

`headalign` = ⟨*empty*⟩ or ⟨*coltype*⟩ — *initially:* ⟨*empty*⟩

This property specifies the alignment for header cells. If left empty, each header cell receives the same alignment as the respective column.

`headbg` = ⟨*color*⟩ — *initially:* `black!14`

This property specifies the background color of the header rows. The ⟨*color*⟩ must be a single color specification that is understood by the xcolor package. The ⟨*color*⟩ is passed directly to the `\rowcolor` macro. If ⟨*color*⟩ is empty, then no background color is produced for the header row.

`headformat` = ⟨*single argument macro*⟩ — *initially:* ⟨*"identity"*⟩

This property specifies a format to be applied to all header cells. The value specified for the `headformat` key is used to format each header. The value can be a macro that takes once argument, through which it is provided the header (as specified in the column's `head` property). Initially, an "identity" macro is used, meaning that each `head` is taken without change.

```
\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 1: table options demo

```
\NewKeyValTable[showrules=false,headbg=blue!25,
   headalign=c,headformat=\textbf,norowbg
 ]{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=headformat, val=\string\textbf}
\Row{opt=norowbg, val=true}
\end{KeyValTable}
```

opt	val
showrules	false
headbg	blue!25
headalign	c
headformat	\textbf
norowbg	true

Figure 1: Examples for table options

rowbg = ⟨*color*⟩ *initially:* white..black!10

This property specifies the background colors of content rows. The ⟨*value*⟩ for this property must be of the format ⟨*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. If ⟨*color*⟩ is empty, then no background color is produced for content rows.

norowbg = true, false *default:* true, *initially:* false

nobg = true, false *default:* true, *initially:* false

These properties are shorthands for rowbg={} (turning off background colors for content rows) and, respectively, for rowbg={},headbg={} (turning off background colors for header rows and for content rows). Using these options without a value is equivalent to using true for the value. For instance, nobg is equivalent to nobg=true.

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

5.2 Column Appearance

Column appearance is configured through the parameters align, head, format, and default of columns in \NewKeyValTable. For the format, the following macro

exists to ensure proper height and depth of rows even if the content itself is more narrow.

`\kvtStrutted[`⟨*inner*⟩`]{`⟨*arg*⟩`}`

This macro places a `\strut` before ⟨*arg*⟩ and a `\strut` after ⟨*arg*⟩. This has the effect that the first and last row of ⟨*arg*⟩ obtain a "natural" height and depth even if their content is smaller. The second `\strut` is omitted when it would cause a new line to be produced. See Section 4 for an example.

5.3 Row Appearance

Through the ⟨*options*⟩ argument of the `\Row` and the `\KeyValRow` macros, the appearance of rows 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 options are supported.

`hidden` = `true`, `false` — *initially:* `false`

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`.

`bg` = ⟨*color*⟩ — *initially:* ⟨*empty*⟩

This property specifies the background color for the particular row. If this option is not specified (or set to an empty value explicitly), the background color is determined by the `rowbg` option of the table.

`above` = ⟨*dimension*⟩ — *initially:* ⟨*empty*⟩
`below` = ⟨*dimension*⟩ — *initially:* ⟨*empty*⟩
`around` = ⟨*dimension*⟩ — *initially:* ⟨*empty*⟩

These properties specify extra vertical space above and, respectively, below the row. The `around` property is a short-hand for setting both, `above` and `below`, to the same value. Note that the vertical 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`.

Initial values for all row options can be set with `\kvtSet{Row/`⟨*option*⟩`=`⟨*value*⟩`}` (see also Section 5.5).

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=Gold]{
  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

5.3.1 Row Styles

Rather than specifying properties for individual rows, keyvaltable also supports named *row styles*.

`style` = ⟨*list of style names*⟩ *initially:* ⟨*empty*⟩

Through this property of rows, a list of styles can be applied to the row. Each style must have been defined with `\kvtNewRowStyle` before.

`\kvtNewRowStyle{`⟨*name*⟩`}{`⟨*row-options*⟩`}`

This macro declares a new row style with the given ⟨*name*⟩ and defines it to be equivalent to using the given ⟨*row-options*⟩. The ⟨*name*⟩ must not already be defined.

`\kvtRenewRowStyle{`⟨*name*⟩`}{`⟨*row-options*⟩`}`

This macro re-defines an existing row style ⟨*name*⟩ with new ⟨*row-options*⟩.

The following example produces the same output as the previous example, but uses row styles.

```
\kvtNewRowStyle{optional}{hidden}
\kvtNewRowStyle{highlight}{above=1ex,bg=Gold}
\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[style=optional]{amount=25g,
  ingredient=cream, step=decorate on top}
\Row[style=highlight]{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

The ⟨*row-options*⟩ in `\kvtNewRowStyle` can be left empty. In this case, the row style does not have any effect on the appearance of rows. However, the style can already be used for "tagging" rows and the final options for the style can be configured at a later point in time.

5.3.2 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

5.4 Cell Appearance

Individual cells can be formatted by using the respective LaTeX code directly in the value of the cell. One can disabled column's configured `format` for the cell by using the starred column name in `\Row`. The following example demonstrates starred column names.

```
\usepackage{url}\urlstyle{sf}
\NewKeyValTable{Links}{
  service;
  url: format=\url }
\begin{KeyValTable}{Links}
  \Row{service=CTAN,
    url=ctan.org/pkg/keyvaltable}
  \Row{service=github,
    url=github.com/Ri-Ga/keyvaltable}
  \Row{service=Google Play, url*=none}
\end{KeyValTable}
```

service	url
CTAN	ctan.org/pkg/keyvaltable
github	github.com/Ri-Ga/keyvaltable
Google Play	none

5.5 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

6 Customizing the Layout

The keyvaltable package provides some means for altering tables beyond those described in the previous sections. Those means are described in the following.

6.1 Custom Table Headers

By default, a table type defined by `\NewKeyValTable` includes a single header row and each column of the table type has a header cell in this row. Through the optional ⟨*layout*⟩ parameter of `\NewKeyValTable`, one can define multiple header rows and can define header cells that span multiple columns.

The following two examples illustrate how the `headers` key in the ⟨*layout*⟩ parameter can be used for specifying custom headers.[1] 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.

[1] In keyvaltable v1.0, the ⟨*layout*⟩ parameter specified *only* the headers and did not use a `headers` key for this. For compatibility, this can be enabled with the `compat=1.0` package option.

```
\NewKeyValTable{Headers1}{
  id:    align=r, default=\thekvtRow.;
  amount: align=r; ingredient: align=l;
  step:  align=X;
}[headers={
    amount+ingredient: head=\textbf{ingredient};
    step: head=\textbf{step}, align=l;
  }
]
\begin{KeyValTable}{Headers1}
\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{Headers2}{
  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;
}[headers={
  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}{Headers2}
\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 a ⟨*value*⟩ of the `headers` key in the ⟨*layout*⟩ parameter is as follows:

- ⟨*value*⟩ 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 properties of the header cell. Supported ⟨*property*⟩ keys are the following.

`align` = ⟨*alignment-letter*⟩, ⟨*empty*⟩ *initially:* `c`

This property specifies the alignment of content in the header cell. The ⟨*value*⟩ can be set to any column alignment understood by the underlying table environment used (see Section 6.4). This particularly includes `l`, `c`, `r`, and `p`, as well as `X` for some of the table environments. The initial value can be modified with `\kvtSet{HeadCell/align=...}`.

`head` = ⟨*text*⟩ *initially:* ⟨*colspec*⟩

This property specifies the content of the header cell. The initial value for this property is the column specification, i.e., "⟨*col*⟩+…+⟨*col*⟩".

6.2 Column Spanning

The keyvaltable package supports column spanning via "column groups". A column group is a collection of adjacent columns, has its own name, and can be assigned a value just like "normal" columns can be. The following example demonstrates how column groups can be defined and be used.

```
\NewKeyValTable{AltRecipe}{
  amount:    align=r, format=\textbf;
  ingredient: align=l;
  step:      align=X;
}[colgroups={
  all: span=step+amount+ingredient
}]
\begin{KeyValTable}{AltRecipe}
\Row{amount=150g, ingredient=ice cream,
  step=put into bowl}
\Row{amount= 50g, ingredient=cherries,
  step=heat up and add to bowl}
\midrule
\Row{all=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		

As the example shows, column groups are defined through the `colgroups` key of the second optional argument of `\NewKeyValTable`. This key expects a semicolon-separated list of individual column groups definitions. Each such definition takes the same shape as a normal column definition – that is, first the name of the column group, then a colon, and then a comma-separated list of column properties. The properties that can be set are the following.

`span` = ⟨*plus-separated columns*⟩

This property specifies which columns the column group shall span, as a plus-separated list of column names. Some or all of the columns can be hidden. All the displayed columns must be adjacent in the table, though.

`align` = ⟨*alignment-letter*⟩, ⟨*empty*⟩ *initially:* `c`

`format` = ⟨*single argument macro*⟩ *initially:* `\kvtStrutted`

These properties are analogous to the respective properties of normal columns. The only difference is that the initial column alignment of column groups is "`c`" while the alignment of normal columns is "`l`".

Initial values for all the `align` and `format` options can be set with `\kvtSet`, via the `ColGroup/align` and, respectively `ColGroup/format` keys (see also Section 5.5).

6.2.1 Manual Column Spanning

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.

6.3 Captions

There are two ways to add captions to (keyvaltable-) tables: The first way is to enclose the table in a table environment. This is particularly suit for tables that do not span multiple pages, such as those produced through the onepage shape (or tabular, tabularx, and tabu – see Section 6.4).

```
\begin{table}
  \begin{KeyValTable}[shape=onepage]{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}
  \caption{Cherries++}
  \label{Cherries}
\end{table}
Table~\ref{Cherries} shows the recipe.
```

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

Table 2: Cherries++

Table 2 shows the recipe.

The second way to add captions is through the caption option of keyvaltable tables. This is particularly suit for tables that can span multiple pages, such as those produced through the multipage shape (or longtable, xltabular, and longtabu – see Section 6.4).

caption = ⟨*text*⟩ *initially:* ⟨*none*⟩
label = ⟨*name*⟩ *initially:* ⟨*none*⟩

These options set the caption and, respectively, label of a table. The caption is added to the end of the table. The following example shows the options in action.

shape	environment	multi-page	caption	X columns	width	packages
onepage	tabular/tabularx			✓	✓	tabularx
multipage	longtabu/xltabular	✓	✓	✓	✓	longtable, xltabular
	with package option compat=1.0:					
onepage	tabu			✓	✓	tabu
multipage	longtabu	✓	✓	✓	✓	tabu, longtable
tabular	tabular					
tabularx	tabularx			✓	✓	tabularx
longtable	longtable	✓	✓			longtable
xltabular	xltabular	✓	✓	✓	✓	xltabular
tabu	tabu			✓	✓	tabu
longtabu	longtabu	✓	✓	✓	✓	tabu, longtable

Table 4: Comparison of table shapes / environments

```
\begin{KeyValTable}[shape=multipage,
  caption=Cherries++, label=Cherries2]{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}
Table~\ref{Cherries2} shows the recipe.
```

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

Table 3: Cherries++

Table 3 shows the recipe.

6.4 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 4 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.

7 Use with Other Packages

7.1 Named References (cleveref)

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[headformat=\textbf]{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

7.2 Tables from CSV Files (datatool and csvsimple)

The keyvaltable package itself does not offer its own functionality for generating tables from CSV files. However, together with existing CSV packages, table content can be sourced from CSV files. The remainder of this section shows how this can be

```
\NewKeyValTable[showrules=false]{ShapeNoX}{
  id: align=l, default=\thekvtTypeRow;
  l: align=l; c: align=c; r: align=r;}[headers={
  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;}[headers={
  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=xltabular]{ShapeWithX}
  \Row{l=left, X=expandable, r=right}
  \Row{l=left-2, X=expandable-2, r=2-right}
\end{KeyValTable}
\begin{KeyValTable}[shape=tabu]{ShapeWithX}
  \Row{l=left, X=expandable, r=right}
  \Row{l=left-2, X=expandable-2, r=2-right}
\end{KeyValTable}
\begin{KeyValTable}[shape=longtabu]{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

xltabular shape

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

tabu shape

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

longtabu shape

id	l	X	r
7	left	expandable	right
8	left-2	expandable-2	2-right

Figure 2: Examples for the shape option

```
snowman,1,carrot,stick into top ball
snowman,2,coffee beans,put diagonally above carrot
cherries,150g,ice cream,put into bowl
cherries,50g,cherries,heat up and add to bowl
```

Listing 1: recipes.csv

datatool The package provides a variety of macros for loading and also displaying CSV database content. The following shows how the macros `\DTLloaddb` and `\DTLforeach*` can be used, together with `\AddKeyValRow` and `\ShowKeyValTable`. The example also shows how a simple filter can be applied to the rows via `\DTLforeach*`.

```
\usepackage{datatool}
\DTLloaddb{recipes}{recipes.csv}
\DTLforeach*[\equal{\Id}{snowman}]{recipes}
  {\Id=id,
   \Amount=amount,\Ingr=ingredient,\Step=step}
  {\AddKeyValRow{Recipe}[expandonce]{
   amount=\Amount,ingredient=\Ingr,step=\Step}}
\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

Two aspects shall be noted. Firstly, we use `\AddKeyValRow` rather than `KeyValTable`, because `\DTLforeach*` interferes with how `KeyValTable` constructs its rows and yields "misplaced `\noalign`" errors. We do not use `\CollectRow` here, because it requires two runs and we do not need the feature to show the table before the rows are specified. Secondly, we use the row option `expandonce` to ensure that the macros `\Amount`, `\Ingr`, and `\Step` are expanded (i.e., replaced by their values). Without this option, all rows would only carry the three macros and display the value that these macros have at the time of the `\ShowKeyValTable`.

expandonce = `true`, `false` — *default:* `true`, *initially:* `false`

expand = `true`, `false` — *default:* `true`, *initially:* `false`

These row options can be used when programmatically constructing the rows of a table, particularly with `KeyValTableContent` and `\CollectRow`. The `expandonce` option expands all the cell values given to a row (default values not included) exactly once before including it in the respective row. The `expand` option fully expands the cell values, in `protect`'ed mode (i.e., robust commands are not expanded).

csvsimple For the sake of our example, using this package is very similar to using datatool.

```
\usepackage{csvsimple}
\csvreader[head to column names,
  filter equal={\id}{cherries}]{recipes.csv}{}
  {\AddKeyValRow{Recipe}[expand]{
     amount=\amount,ingredient=\ingredient,
     step=\step}}
\ShowKeyValTable{Recipe}
```

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

Two differences are noteworthy here: First, we can avoid specifying macro names for the columns through the `head to column names`, which uses the column names

as macro names. Second, we have to use the expand option rather than expandonce here, because csvsimple apparently does not directly store the column value in the respective macro.

7.3 Computational Cells (xint)

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

7.4 Cell Formatting (makecell)

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

8 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.

tablestyles: This package simplifies typesetting tables with common and/or more appealing appearances than default LaTeX tables. This corresponds to what keyvaltable supports with the various coloring and formatting options to `\kvtSet`, `\NewKeyValTable`, and individual tables. The tablestyles package builds on the default LaTeX environments and syntax for typesetting tables (with column alignments specified in an argument to the table environment, and columns separated by `&` in the body of the environment).

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.

9 Future Work

- support for different headers on the first page vs. on subsequent pages of a multipage table; support configurable spacing between and above/below header rows
- support for more flexibility with regards to captions position (top vs. bottom) and distinct captions on first/middle/last page of the table.
- improved row coloring that makes sure that the alternation re-starts on continued pages of a table that spans several pages
- rerun detection for recorded rows (possibly via rerunfilecheck)
- nesting of `KeyValTable` environments (this is so far not tested by the package author and might not work or work only to a limited extent)

10 Implementation

Content

10.1 Package Dependencies . 22
10.2 Auxiliary Code 22
10.3 Setting Options 23
10.4 Declaring Key-Value Tables 26
10.5 Custom Layout Parameters 29
10.6 Row Numbering and Labeling 34
10.7 Key-Value Table Content 35
10.8 Collecting Key-Value Table Content 44
10.9 Package Options 48
10.10 Compatibility 48

10.1 Package Dependencies

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}
```

10.2 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` These macros produce error and warning messages.

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

`\kvt@setkeys` `\kvt@setcmdkeys` `\kvt@setcskeys` The `\kvt@setkeys{⟨keys⟩}{⟨fam⟩}` macro abbreviates `\setkeys[kvt]⟨fam⟩⟨keys⟩` (note the reverse order of arguments). The `\kvt@setcmdkeys{⟨keycmd⟩}{⟨fam⟩}` and `\kvt@setcskeys{⟨keycs⟩}{⟨fam⟩}` abbreviate the cases where ⟨keys⟩ are stored in macro ⟨keycmd⟩ or, respectively, stored in a macro with name ⟨keycs⟩.

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

```
14   \expandafter\kvt@setkeys\expandafter{#1}{#2}}
15 \newcommand\kvt@setcskeys[2]{%
16   \expandafter\kvt@setcmdkeys\expandafter{\csname #1\endcsname}{#2}}
```

\kvt@colsetkeys
\kvt@colsetcmdkeys
\kvt@colsetcskeys

The \kvt@colsetkeys{⟨*fam*⟩}{⟨*keys*⟩} macro abbreviates \setkeys[KeyValTable]⟨*fam*⟩⟨*keys*⟩. The \kvt@colsetcmdkeys{⟨*famcmd*⟩}{⟨*keys*⟩} and \kvt@colsetcskeys{⟨*famcs*⟩}{⟨*keys*⟩} abbreviate the cases where ⟨*fam*⟩ is stored in macro ⟨*famcmd*⟩ or, respectively, stored in a macro with name ⟨*famcs*⟩.

```
17 \newcommand\kvt@colsetkeys[2]{\setkeys[KeyValTable]{#1}{#2}}
18 \newcommand\kvt@colsetcmdkeys[2]{%
19   \expandafter\kvt@colsetkeys\expandafter{#1}{#2}}
20 \newcommand\kvt@colsetcskeys[2]{%
21   \expandafter\kvt@colsetcmdkeys\expandafter{\csname #1\endcsname}{#2}}
```

\kvtStrutted

The \kvtStrutted[⟨*inner*⟩]{⟨*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. The optional [⟨*inner*⟩] argument, if provided, should be a macro that takes one argument. In this case, instead of ⟨*arg*⟩, ⟨*inner*⟩{⟨*arg*⟩} is prefixed and sufficed with \strut.

```
22 \newcommand\kvtStrutted[2][\@firstofone]{%
23   \strut#1{#2}\ifhmode\expandafter\strut\fi}
```

10.3 Setting Options

\kvtSet

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

```
24 \newcommand\kvtSet[1]{%
25   \kvt@setkeys{#1}{global,Table,Column}%
26   \ifdefvoid\kvt@@presetqueue{}
27     {\kvt@@presetqueue\undef\kvt@@presetqueue}}
```

\kvt@lazypreset

The \kvt@@lazypreset{⟨*family*⟩}{⟨*head keys*⟩} macro collects a request for presetting ⟨*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.

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

\kvt@keysetter

The \kvt@keysetter{⟨*macro*⟩}{⟨*fam*⟩}{⟨*key*⟩}{⟨*value*⟩}{⟨*func*⟩} macro is an auxiliary macro that can be used inside the "func" argument of \define@...key macros. If ⟨*macro*⟩ is not defined, \kvt@keysetter expands to an instance of \kvt@lazypreset in order to set a global default. Otherwise, \kvt@keysetter expands to ⟨*func*⟩, which is supposed to set a key for the specific context referenced by ⟨*macro*⟩.

```
30 \newcommand\kvt@keysetter[5]{%
31   \ifdefvoid{#1}
32     {\kvt@lazypreset{#2}{#3=#4}}
33     {#5}}
```

\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}
```

10.3.1 Table Options

The following code defines the possible table options.

```
\define@cmdkey[kvt]{Table}{rowbg}{}
\define@cmdkey[kvt]{Table}{headbg}{}
\define@boolkey[kvt]{Table}{norowbg}[true]{%
  \kvt@setkeys{rowbg={}}{Table}}
\define@key[kvt]{Table}{nobg}[true]{%
  \kvt@setkeys{rowbg={},headbg={}}{Table}}
\define@cmdkey[kvt]{Table}{headalign}{}
\define@cmdkey[kvt]{Table}{headformat}{}
\define@cmdkey[kvt]{Table}{width}{}
\define@boolkey[kvt]{Table}{showhead}{}
\define@boolkey[kvt]{Table}{showrules}{}
\define@cmdkey[kvt]{Table}{caption}{}
\define@cmdkey[kvt]{Table}{label}{}

```

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

```
\define@choicekey[kvt]{Table}{shape}
  {multipage,onepage,tabular,longtable,tabularx,xltabular,tabu,longtabu}
  {\csdef{cmdkvt@Table@shape}{#1}}
```

10.3.2 Column Options

The following code defines the possible column options.

```
\define@key[kvt]{Column}{default}{\kvt@colkeysetter{default}{#1}}
\define@key[kvt]{Column}{format}{\kvt@colkeysetter{format}{#1}}
\define@key[kvt]{Column}{align}{\kvt@colkeysetter{align}{#1}}
\define@key[kvt]{Column}{head}{\kvt@colkeysetter{head}{#1}}
\define@boolkey[kvt]{Column}{hidden}[true]{%
  \kvt@colkeysetter{hidden}{#1}}
```

\kvt@colkeysetter The \kvt@colkeysetter{⟨*key*⟩}{⟨*value*⟩} specializes \kvt@keysetter for column options.

```
\newcommand\kvt@colkeysetter[2]{%
  \kvt@keysetter{\kvt@@column}{Column}{#1}{#2}{%
    \csdef{kvt@col@#1@\kvt@@column}{#2}}}
```

\kvt@def@globalopt \kvt@def@globalopts The \kvt@def@globalopt{⟨*family*⟩}key macro creates an option key "⟨*family*⟩/⟨*key*⟩" for \kvtSet that set the preset value for the ⟨*key*⟩ in ⟨*family*⟩. The \kvt@def@globalopts{⟨*family*⟩}keys macro extends the former macro to comma-separated lists of ⟨*keys*⟩ within a single ⟨*family*⟩.

```
\newcommand\kvt@def@globalopt[2]{%
  \define@key[kvt]{global}{#1/#2}{\kvt@lazypreset{#1}{#2={##1}}}}
```

```
\newcommand\kvt@def@globalopts[2]{%
  \forcsvlist{\kvt@def@globalopt{#1}}{#2}}

\define@cmdkey[kvt]{ColGroup}{span}{%
  \csdef{kvt@colgrp@span@\kvt@@colgrp}{#1}}
\define@cmdkey[kvt]{ColGroup}{align}{%
  \csdef{kvt@colgrp@align@\kvt@@colgrp}{#1}}
\define@cmdkey[kvt]{ColGroup}{format}{%
  \csdef{kvt@colgrp@format@\kvt@@colgrp}{#1}}
\kvt@def@globalopts{ColGroup}{align, format}
```

10.3.3 Layout Customization Options

The following defines the options for the second optional argument to `\NewKeyValTable`. These options intentionally do not support setting global defaults via `\kvtSet`.

```
\define@cmdkey[kvt]{Layout}{headers}{%
  \expandafter\kvt@parseheadrows\expandafter{\kvt@@tname}{#1}}
\define@cmdkey[kvt]{Layout}{colgroups}{%
  \expandafter\kvt@parsecolgroups\expandafter{\kvt@@tname}{#1}}
```

The following defines the options for header cells.

```
\define@key[kvt]{HeadCell}{head}{%
  \csdef{kvt@@hdcell@head@\kvt@@hdcell}{#1}}
\define@key[kvt]{HeadCell}{align}{%
  \csdef{kvt@@hdcell@align@\kvt@@hdcell}{#1}}
\kvt@def@globalopts{HeadCell}{align}
```

10.3.4 Row Options

The following block declares the known row options. Note that these are not enabled for `\kvtSet`.

```
\define@cmdkey[kvt]{Row}{bg}{}
\define@boolkey[kvt]{Row}{hidden}[true]{}
\define@cmdkey[kvt]{Row}{below}{}
\define@cmdkey[kvt]{Row}{above}{}
\define@key[kvt]{Row}{around}{%
  \kvt@setkeys{below={#1},above={#1}}{Row}}
\define@key[kvt]{Row}{style}{\kvt@UseRowStyles{#1}}
\define@boolkey[kvt]{Row}{uncounted}[true]{}
\define@boolkey[kvt]{Row}{expand}[true]{}
\define@boolkey[kvt]{Row}{expandonce}[true]{}
\kvt@def@globalopts{Row}{
  bg,hidden,below,above,around,style,uncounted,
  expand,expandonce}
```

10.3.5 Option Defaults

The following sets the default values for the options.

```
\kvtSet{%
  rowbg=white..black!10,
  headbg=black!14,
```

```
  showhead=true,
  showrules=true,
  headformat=\@firstofone,
  headalign=,
  shape=multipage,
  width=\linewidth,
  caption={}, label={},
```

Column options

```
  default=,
  format=\kvtStrutted,
  align=l,
  head=,
  hidden=false,
  Row/bg={},
  Row/hidden=false,
  Row/above={},
  Row/below={},
  Row/uncounted=false,
  Row/expand=false,
  Row/expandonce=false,
  ColGroup/align=c,
  ColGroup/format=\kvtStrutted,
  HeadCell/align=c,
}
```

10.4 Declaring Key-Value Tables

\NewKeyValTable

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

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

Before doing anything, check whether ⟨*tname*⟩ has already been defined.

```
  \ifinlist{#2}{\kvt@alltables}
    {\kvt@error{Table type with name '#2' already defined}
      {Check '#2' for typos and check other uses of
      \string\NewKeyValTable}}{}%
```

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@allcolumns@#2}{}%
\csdef{kvt@displaycols@#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}%
```

By default, a single header row is constructed.

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

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

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

Finally, parse ⟨*layout*⟩.

```
\kvt@parselayout{#4}{#2}%
}
```

\kvt@parsecolspec 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{#2}%
  \trim@spaces@in\kvt@@column
  \expandafter\kvt@parsecolspec@i\expandafter{\kvt@@column}{#1}{#3}}
\newcommand\kvt@parsecolspec@i[3]{\kvt@parsecolspec@ii{#2}{#1}{#3}}
\newcommand\kvt@parsecolspec@ii[3]{%
  \def\kvt@@column{#1@#2}%
```

Check and record the column name first.

```
  \ifinlistcs{#2}{kvt@allcolumns@#1}
    {\kvt@error{Column name '#2' declared more than once in table type
      '#1'}{Check '#2' for typos; column names declared so far:%
      \forlistcsloop{ }{kvt@allcolumns@#1}}}{}%
  \listcsadd{kvt@allcolumns@#1}{#2}%
  \kvt@setkeys{#3}{Column}%
```

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 header. 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@displaycols@#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. The starred variant of the key disables the column's format for the cell.

```
  \define@cmdkey[KeyValTable]{#1}{#2}[]{}%
  \define@key[KeyValTable]{#1}{#2*}{%
    \csdef{cmdKeyValTable@#1@#2}{##1}%
    \csdef{kvt@@noformat@#1@#2}{1}}%
  \presetkeys[KeyValTable]{#1}{#2}{}%
```

The \kvt@parsecolspec macro is not necessarily enclosed in a group. To avoid leaking a local \kvt@@column value to the outer (global) scope, we explicitly undefine it.

```
  \undef\kvt@@column}
```

\kvt@defaultheader

The \kvt@defaultheader{⟨*head1*⟩}…{⟨*headn*⟩}\@nil macro, takes n header cell titles, ⟨*head1*⟩ to ⟨*headn*⟩ and formats them based on the headformat 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\kvt@rowcolorornot{\cmdkvt@Table@headbg}%
  \kvt@defaultheader@i}
\newcommand\kvt@defaultheader@i[1]{%
  \kvt@ifnil{#1}{\noexpand\tabularnewline}{%
    \unexpanded{&}%
    \ifdefvoid\cmdkvt@Table@headalign
      {\expandonce\cmdkvt@Table@headformat{\unexpanded{#1}}}
      {\noexpand\multicolumn{1}{\expandonce\cmdkvt@Table@headalign}
        {\expandonce\cmdkvt@Table@headformat{\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{}
```

10.5 Custom Layout Parameters

\kvt@parselayout The \kvt@parselayout{⟨*layout-opts*⟩}{⟨*tname*⟩} macro parses the layout options, ⟨*layout-opts*⟩, for table type ⟨*tname*⟩,

```
\newcommand\kvt@parselayout[2]{%
  \def\kvt@@tname{#2}%
```

Now parse the ⟨*layout-opts*⟩. The keys are defined such that their handlers already do the parsing.

```
  \kvt@setkeys{#1}{Layout}%
  \undef\kvt@@tname}
```

\kvt@parsecolgroups The \kvt@parsecolgroups{⟨*tname*⟩}{⟨*spec*⟩} macro parses the specification, ⟨*spec*⟩, of column groups for table type ⟨*tname*⟩.

```
\newcommand\kvt@parsecolgroups[2]{%
  \begingroup
```

\kvt@@result collects the parsing outcome code that shall escape the group started above.

```
  \def\kvt@@result{}%
  \def\do##1{\kvt@parsecolgroup{#1}##1::\@undefined}%
  \kvt@dossvlist{#2}%
  \expandafter\endgroup\kvt@@result}
```

The \kvt@parsecolgroup{⟨*tname*⟩}{⟨*cgname*⟩}{⟨*cgopts*⟩}{⟨*empty*⟩} macro parses a single column group, ⟨*cgname*⟩ with options ⟨*cgopts*⟩.

```
\def\kvt@parsecolgroup#1#2:#3:#4\@undefined{%
  \ifinlistcs{#2}{kvt@allcolumns@#1}{\kvt@error
    {Name `#2' cannot be used for a column group in table type `#1',
     as it is already used for a column}
    {Check the \string\NewKeyValTable{#1} for
     the names of known columns and check `#2' for a typo.}}{}%
  \ifinlistcs{#2}{kvt@grpcolkeys@#1}{\kvt@error
    {Name `#2' is used twice in table type `#1'}
    {Check the \string\NewKeyValTable{#1} for typos in the names of
     columns groups.}}{}%
  \def\kvt@@colgrp{#2}%
  \kvt@setkeys{#3}{ColGroup}%
  \kvt@checkcolgroupcs{kvt@colgrp@span@\kvt@@colgrp}{#1}{#2}%
  \eappto\kvt@@result{%
```

The following defines the \Row key for ⟨*cgname*⟩, as an abbreviation for setting the value of the first displayed column of ⟨*cgname*⟩ (\kvt@@colgrp@first to a \multicolumn that spans the "right" number of columns. Notice the "*" after \kvt@@colgrp@first, which disables the first column's default formatting to replace it by the formatting of ⟨*cgname*⟩.

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

The following \ifdefvoid check ensures that if ⟨*cgname*⟩ is a hidden column group (i.e., a column group of which all spanned columns are hidden), then setting ⟨*cgname*⟩ to a value has no effect.

```
  \ifdefvoid\kvt@@colgrp@first{}{%
    \noexpand\setkeys[KeyValTable]{#1}{%
      \expandonce\kvt@@colgrp@first*=\noexpand\multicolumn
        {\expandonce\kvt@@colgrp@n}%
        {\csexpandonce{kvt@colgrp@align@#2}}%
        {\csexpandonce{kvt@colgrp@format@#2}{\unexpanded{##1}}}}%
  }%
}%
\noexpand\listcsadd{kvt@grpcolkeys@#1}{#2}}}
```

\kvt@checkcolgroup The \kvt@checkcolgroup{⟨*span-psv*⟩}{⟨*tname*⟩}{⟨*cgname*⟩} macro performs some checks on ⟨*span-psv*⟩ as a specification of which columns shall be spanned by a group column of name ⟨*cgname*⟩. The checks are

- whether all column names are indeed columns of ⟨*tname*⟩,
- whether each column appears at most once in the column group, and
- whether the (displayed) columns from ⟨*span-psv*⟩ appear consecutively in ⟨*tname*⟩.

The macro returns the number of spanned (displayed!) columns in \kvt@@colgrp@n and the name of the first column in \kvt@@colgrp@first.

Fixme: There can probably be some code sharing with \kvt@parseheadrow and \kvt@parsecolgroup.

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

First, check individual colums in ⟨*span-psv*⟩ and transfer them into a "map", kvt@@incolgrp@ that simply records which column names occur in ⟨*span-psv*⟩.

```
  \def\kvt@@psvdo##1{%
    \ifinlistcs{##1}{kvt@allcolumns@#2}{}{\kvt@error
      {Column `##1' referenced in column group `#3' not known
       in table type `#2'}
      {Check the \string\NewKeyValTable{#2} for
       the names of known columns and check `##1' for a typo.}}%
    \ifcsvoid{kvt@@incolgrp@##1}{}{\kvt@error
      {Column `##1' used more than once in column group `#3' of table
       type `#2'}
      {Check `##1' for a typo.}}%
    \csdef{kvt@@incolgrp@##1}{#2}%
  }\kvt@forpsvlist{\kvt@@psvdo}{#1}%
```

The following two macros are the "return values".

```
\def\kvt@@colgrp@n{0}%
\let\kvt@@colgrp@first\relax
```

Second, iterate over the displayed columns of ⟨*tname*⟩ to check whether the columns in ⟨*span-psv*⟩ are consecutive. For this, use `\kvt@@status` to track whether no column of ⟨*span-psv*⟩ has yet been visited (value 0, the initial value), whether the current column is part of ⟨*span-psv*⟩ (value 1), and whether columns of ⟨*span-psv*⟩ have been visited but the current column is not part of ⟨*span-psv*⟩ (value 2).

```
\def\kvt@@status{0}%
```

`\kvt@@coldo{`⟨*column*⟩`}` is applied to each displayed column, in order.

```
\def\kvt@@coldo##1{%
  \ifcsvoid{kvt@@incolgrp@##1}
```

If ⟨*column*⟩ is *not* in ⟨*span-psv*⟩, then change `\kvt@@status` from 1 to 2, but do not change it when it is 0 or 2.

```
    {\expandafter\ifcase\kvt@@status \or
      \def\kvt@@status{2}\fi}%
```

If ⟨*column*⟩ is in ⟨*span-psv*⟩, then change `\kvt@@status` from 0 to 1 and record ⟨*column*⟩ as `\kvt@@colgrp@first`; if `\kvt@@status` is previously 2, then the columns in ⟨*span-psv*⟩ would not be consecutively displayed and, hence, an error is raised.

```
    {\expandafter\ifcase\kvt@@status
      \def\kvt@@status{1}\def\kvt@@colgrp@first{##1}%
      \or\or
      \kvt@error{Column group `\kvt@@colgrp' must consist of only
         consecutive columns, but it is not}%
        {Compare `\string\kvt@@curgrp' to the column ordering as
         specified in `\string\NewKeyValTable{#1}'}%
      \fi
      \edef\kvt@@colgrp@n{\the\numexpr\kvt@@colgrp@n+1\relax}}%
}\forlistcsloop{\kvt@@coldo}{kvt@displaycols@#2}}
```

`\kvt@checkcolgroupcs` The `\kvt@checkcolgroupcs{`⟨*span-psv-cs*⟩`}{`⟨*tname*⟩`}{`⟨*cgname*⟩`}` macro is the same as `\kvt@checkcolgroup` except that it takes a control sequence name as its first argument rather than a plus-separated list directly.

```
\newcommand\kvt@checkcolgroupcs[3]{%
  \expandafter\expandafter\expandafter
  \kvt@checkcolgroup
  \expandafter\expandafter\expandafter{\csname #1\endcsname}{#2}{#3}}
```

`\kvt@parseheadrows` The `\kvt@parseheadrows{`⟨*tname*⟩`}{`⟨*headers*⟩`}` macro parses the values of the `headers` key in the ⟨*layout*⟩ argument of `\NewKeyValTable`. The values are `\\`-separated lists of header rows, and the rows are semicolon-separated lists of header cells. Each header cell can span zero, one, or more visible columns. If the `headers` key is not set (or empty), then the default header (based on the column specification alone) is used, as set by `\kvt@NewKeyValTable`.

```
\newcommand\kvt@parseheadrows[2]{%
  \ifstrempty{#2}{}{\kvt@parseheadrows@i{#2}{#1}}}
\newcommand\kvt@parseheadrows@i[2]{%
```

```
\csdef{kvt@@custheadrows@#2}{}%
\csdef{kvt@headrowcount@#2}{0}%
\begingroup
\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@@custheadrows@#2}{%
         \csexpandonce{kvt@headings@#2}}}}
    {\appto\kvt@@parseheadrows{\kvt@parseheadrow{#2}{##1}}}%
```

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

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

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

```
\expandafter\endgroup\kvt@@parseheadrows
\csletcs{kvt@headings@#2}{kvt@@custheadrows@#2}}
```

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

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

First parse ⟨*colspec*⟩, populating the \kvt@@hdcellof@⟨*colname*⟩ macros that associate each column with the header cell to which the column belongs (in this row).

```
\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@@curhd and \kvt@@lasthd hold the name of the header cell in which the current column and, respectively, previous column are in. Each of the two macros is undefined if there is no such header cell.

```
\let\kvt@@tmpgrphd\@empty
\kvt@@span\z@
\undef\kvt@@curhd \undef\kvt@@lasthd
```

Next, loop over all displayed (non-hidden) columns stored in \kvt@displaycols@⟨*tname*⟩. The following \do{⟨*colname*⟩} macro 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@@curhd{kvt@@hdcellof@##1}%
  \ifdefequal\kvt@@curhd\kvt@@lasthd
```

If the header cell has not changed, simply increase the spanning counter.

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

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

```
      {\ifnum\kvt@@span>\z@ \expandafter\kvt@concludecolumn\fi
       \ifdefvoid\kvt@@curhd{}{\ifcsdef{kvt@@hdcelldone@\kvt@@curhd}{%
         \kvt@error{Header cell `\kvt@@curhd' must consist of only
            consecutive columns, but it is not}%
           {Compare `\string\kvt@@curhd' to the column ordering as
           specified in `\string\NewKeyValTable{#1}'}}{}}%
       \kvt@@span\@ne \let\kvt@@lasthd\kvt@@curhd}%
  }\dolistcsloop{kvt@displaycols@#1}%
  \kvt@concludecolumn
```

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

```
  \appto\kvt@@tmpgrphd{\tabularnewline}%
  \edef\do{\noexpand\csappto{kvt@@custheadrows@#1}{%
    \unexpanded{\noexpand\kvt@rowcolorornot{\cmdkvt@Table@headbg}}%
    \noexpand\unexpanded{\expandonce{\kvt@@tmpgrphd}}}}%
  \expandafter\endgroup\do}
```

\kvt@rowcolorornot The \kvt@rowcolorornot{⟨*color*⟩} expands to \rowcolor{⟨*color*⟩} if ⟨*color*⟩ is nonempty and does have no effect if ⟨*color*⟩ is empty.

```
\newcommand\kvt@rowcolorornot[1]{\ifstrempty{#1}{}{\rowcolor{#1}}}
```

\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 header cell 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@@lasthd{}{%
    \eappto\kvt@@tmpgrphd{\noexpand\multicolumn
      {\the\kvt@@span}
      {\expandonce\kvt@@extraalign
       \csexpandonce{kvt@@hdcell@align@\kvt@@lasthd}}
      {\csexpandonce{kvt@@hdcell@head@\kvt@@lasthd}}}%
```

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

```
\cslet{kvt@@hdcelldone@\kvt@@lasthd}{\@ne}}}
```

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

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

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

```
  \def\kvt@@colreg##1{%
    \ifinlistcs{##1}{kvt@allcolumns@#1}{}
      {\kvt@error{Column `##1', referenced in header cell `#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@@hdcellof@##1}
      {\kvt@error{Column `##1' used in more than one header cell}
         {Check the fourth, optional argument of \string\NewKeyValTable
         and eliminate multiple occurrences of column `##1'.}}
      {\csdef{kvt@@hdcellof@##1}{#2}}%
  }\kvt@forpsvlist{\kvt@@colreg}{#2}%
```

Now parse the ⟨*config*⟩ of the header cell.

```
  \def\kvt@@hdcell{#2}%
  \kvt@setkeys{#3}{HeadCell}}
```

10.6 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}}%
  \addtocounter{kvt@LabelCtr}{-1}%
  \refstepcounter{kvt@LabelCtr}%
```

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

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

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

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

10.7 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][]{%
```

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

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

  \kvt@SetOptions{#2}{#1}%
  \csuse{kvt@StartTable@\cmdkvt@Table@shape}{#2}%
}{%
  \csuse{kvt@EndTable@\cmdkvt@Table@shape}}
```

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

```
\newcommand\kvt@SetOptions[2]{%
  \begingroup\edef\kvt@@do{\endgroup\noexpand%
    \kvt@setkeys{\csexpandonce{kvt@options@#1},\unexpanded{#2}}{Table}%
  }\kvt@@do}
```

10.7.1 Table Environment Code

\kvt@StartTabularlike The \kvt@StartTabularlike{⟨*env*⟩}{⟨*tname*⟩} macro begins a table environment for the given table type ⟨*tname*⟩. The ⟨*env*⟩ parameter specifies the concrete environment name.

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

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

```
  \gdef\kvt@@recenttable{#2}%
  \metatblAtEnd{#1}{\kvt@@endhook}\let\kvt@@endhook\relax%
  \ifbool{kvt@Table@showrules}
    {\def\kvt@@rule##1{\csuse{##1rule}}}
    {\def\kvt@@rule##1{}}%
  \appto\kvt@@endhook{\kvt@@rule{bottom}}
```

Adding caption and label, if given, to the end hook. This displays the caption solely at the very end of the table.

```
  \ifdefempty\cmdkvt@Table@caption{}{%
    \metatblHasCaption{#1}
      {\appto\kvt@@endhook{\rowcolor{white}%
        \caption{\cmdkvt@Table@caption}}%
       \ifdefempty\cmdkvt@Table@label{}{%
         \appto\kvt@@endhook{\expandafter%
           \label\expandafter{\cmdkvt@Table@label}}}}
      {\kvt@warn{Caption lost, table environment '#1'
                 does not support captions.}}}%
```

Initializing the row counters. The global counter kvtTotalRow needs no local initialization.

```
  \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.

```
  \begingroup\edef\kvt@@do{\endgroup
    \metatblIsTabu{#1}{}{\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
      \metatblHasWidth{#1}
        {\metatblIsTabu{#1}
          {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}}
      {}%
    \metatblIsTabu{#1}
```

```
      {\noexpand\kvt@taburowcolors{\expandonce\cmdkvt@Table@rowbg}}{}%
    \metatblIsLong{#1}{\noexpand\endhead}{}%
  }\kvt@@do}
```

\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. If ⟨*colors*⟩ is empty, then no row colors are setup.

```
\newcommand\kvt@dottedrowcolors[2]{%
  \ifstrempty{#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}}}
```

\kvt@taburowcolors The \kvt@taburowcolors{⟨*colors*⟩} expands to \taburowcolors{⟨*colors*⟩} if ⟨*colors*⟩ is nonempty and does have no effect if ⟨*color*⟩ is empty.

```
\newcommand\kvt@taburowcolors[1]{%
  \ifstrempty{#1}{}{\taburowcolors{#1}}}
```

\kvt@DefineStdTabEnv The \kvt@DefineStdTabEnv[⟨*shape*⟩]{⟨*env*⟩} 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.

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[2][]{%
  \expandafter\newcommand\csname kvt@StartTable@#1\endcsname[1]{%
    \kvt@StartTabularlike{#2}{##1}}%
  \csedef{kvt@EndTable@#1}{%
    \expandafter\noexpand\csname end#2\endcsname}}
```

\kvt@DefineDualTabEnv The \kvt@DefineDualTabEnv{⟨*shape*⟩}{⟨*nonX-env*⟩}{⟨*X-env*⟩} macro defines the macros for the given ⟨*shape*⟩ name. The macros are defined in a way such that the table environment ⟨*nonX-env*⟩ is used for typesetting tables that do not use X columns and that table environment ⟨*X-env*⟩ is used for typesetting tables that do use X columns.

```
\newcommand\kvt@DefineDualTabEnv[3]{%
  \expandafter\newcommand\csname kvt@StartTable@#1\endcsname[1]{%
    \kvt@ifhasXcolumns{##1}
      {\csedef{kvt@EndTable@#1}{%
         \expandafter\noexpand\csname end#3\endcsname}%
       \kvt@StartTabularlike{#3}{##1}%
      }{\csedef{kvt@EndTable@#1}{%
```

```
413           \expandafter\noexpand\csname end#2\endcsname}%
414         \kvt@StartTabularlike{#2}{##1}}}}
```

\kvt@ifhasXcolumns The \kvt@ifhasXcolumns{⟨*tname*⟩}{⟨*iftrue*⟩}{⟨*iffalse*⟩} takes a table type ⟨*tname*⟩ and checks whether the table type contains an "X" column. If such a column is contained, the macro expands to ⟨*iftrue*⟩. Otherwise, it expands to ⟨*iffalse*⟩.

```
415 \newcommand\kvt@ifhasXcolumns[1]{%
416   \expandafter\expandafter\expandafter\metatbl@ifhasXcolumns
417   \expandafter\expandafter\expandafter{%
418     \csname kvt@alignments@#1\endcsname}}
```

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

```
419 \kvt@DefineStdTabEnv{tabular}
420 \kvt@DefineStdTabEnv{longtable}
421 \kvt@DefineStdTabEnv{tabularx}
422 \kvt@DefineStdTabEnv{xltabular}
423 \kvt@DefineStdTabEnv{tabu}
424 \kvt@DefineStdTabEnv{longtabu}
```

10.7.2 Table Environment Properties

The following code maintains properties about known table environments. This code does not depend on other code of the keyvaltable package but is only used by keyvaltable.

The following properties can be maintained about table environments.

```
425 \define@boolkey[metatbl]{EnvProp}{isLong}{\metatbl@boolprop{isLong}{#1}}
426 \define@boolkey[metatbl]{EnvProp}{isTabu}{\metatbl@boolprop{isTabu}{#1}}
427 \define@boolkey[metatbl]{EnvProp}{hasWidth}{%
428   \metatbl@boolprop{hasWidth}{#1}}
429 \define@boolkey[metatbl]{EnvProp}{hasCaption}{%
430   \metatbl@boolprop{hasCaption}{#1}}
431 \define@cmdkey[metatbl]{EnvProp}{packages}{\metatbl@setprop{pkg}{#1}}
```

The `atEnd` property shall be set to TEX code with one argument (i.e., using the positional argument `#1`) that adds its argument to the end of the active table environment's final content. Finding such code is not obvious for table environments that collect the content of the environment, like `tabularx` does, for instance.

```
432 \define@key[metatbl]{EnvProp}{atEnd}{\metatbl@setprop[1]{atEnd}{#1}}
```

\metatblRegisterEnv The \metatblRegisterEnv{⟨*env-name*⟩}{⟨*properties*⟩} macro registers a table environment with name ⟨*env-name*⟩ and sets its properties according to ⟨*properties*⟩, a comma-separated key-value list.

```
433 \newrobustcmd\metatblRegisterEnv[2]{%
434   \edef\metatbl@@envname{#1}%
435   \setkeys[metatbl]{EnvProp}{#2}}
```

\metatbl@setprop The \metatbl@setprop[⟨*n*⟩]{⟨*key*⟩}{⟨*value*⟩} macro defines a macro with ⟨*n*⟩ arguments (0 by default) for the environment stored in \metatbl@@envname and the given ⟨*key*⟩. This macro then expands to ⟨*value*⟩.

```
436 \newcommand\metatbl@setprop[3][0]{%
```

```
    \expandafter\newcommand
      \csname metatbl@EnvProp@#2@\metatbl@@envname\endcsname[#1]{#3}}
```

\metatbl@boolprop

The \metatbl@boolprop{⟨*prop*⟩}{⟨*value*⟩} macro stores the Boolean value ⟨*value*⟩ in a property ⟨*prop*⟩ for the environment stored in \metatbl@@envname.

```
\newcommand\metatbl@boolprop[2]{%
  \providebool{metatbl@EnvProp@#1@\metatbl@@envname}%
  \setbool{metatbl@EnvProp@#1@\metatbl@@envname}{#2}}
```

\metatblIsLong
\metatblIsTabu
\metatblHasWidth
\metatblHasCaption

The macro \metatblIsLong{⟨*env-name*⟩}{⟨*iftrue*⟩}{⟨*iffalse*⟩} expands to ⟨*iftrue*⟩ if ⟨*env-name*⟩ is a "long" table environment, i.e., one that can span multiple pages. Otherwise, the macro expands to ⟨*iffalse*⟩. The macro \metatblIsTabu{⟨*env-name*⟩}{⟨*iftrue*⟩}{⟨*iffalse*⟩} expands to ⟨*iftrue*⟩ if ⟨*env-name*⟩ is a table environment that inherits from tabu and expands to ⟨*iffalse*⟩ otherwise. The macro \metatblHasWidth{⟨*env-name*⟩}{⟨*iftrue*⟩}{⟨*iffalse*⟩} expands to ⟨*iftrue*⟩ if ⟨*env-name*⟩ is a table environment that expects a width argument and expands to ⟨*iffalse*⟩ otherwise. \metatblHasCaption{⟨*env-name*⟩}{⟨*iftrue*⟩}{⟨*iffalse*⟩} expands to ⟨*iftrue*⟩ if ⟨*env-name*⟩ is a table environment that supports a caption and expands to ⟨*iffalse*⟩ otherwise.

```
\newcommand\metatblIsLong[1]{\ifbool{metatbl@EnvProp@isLong@#1}}
\newcommand\metatblIsTabu[1]{\ifbool{metatbl@EnvProp@isTabu@#1}}
\newcommand\metatblHasWidth[1]{\ifbool{metatbl@EnvProp@hasWidth@#1}}
\newcommand\metatblHasCaption[1]{\ifbool{metatbl@EnvProp@hasCaption@#1}}
```

\metatblUsePackage
\metatblRequire

The \metatblUsePackage{⟨*env-names*⟩} and \metatblRequire{⟨*env-names*⟩} macros load the packages required for typesetting KeyValTable tables based on the table environments listed in ⟨*env-names*⟩. The former aims more at normal document use, the second at use by package developers.

```
\newcommand\metatblUsePackage[1]{%
  \def\do##1{\metatbl@csnamearg\usepackage{metatbl@EnvProp@pkg@##1}}%
  \docsvlist{#1}}
\newcommand\metatblRequire[1]{%
  \def\do##1{\metatbl@csnamearg\RequirePackage{metatbl@EnvProp@pkg@##1}}%
  \docsvlist{#1}}
```

\metatblAtEnd

The \metatblAtEnd{⟨*env-name*⟩}{⟨*code*⟩} macro registers ⟨*code*⟩ for addition at the end of tables based on the ⟨*env-name*⟩ environment.

```
\newcommand\metatblAtEnd[2]{% #1=env-name, #2=code
  \csname metatbl@EnvProp@atEnd@#1\endcsname{#2}}
```

\metatbl@csnamearg

The auxiliary macro \metatbl@csnamearg{⟨*command*⟩}{⟨*csname*⟩} passes the expansion of the macro with name ⟨*csname*⟩ as the first argument to ⟨*command*⟩.

```
\newcommand\metatbl@csnamearg[2]{%
  \expandafter\expandafter\expandafter#1%
  \expandafter\expandafter\expandafter{\csname#2\endcsname}}
```

The following are the properties of some basic table environments.

```
\metatblRegisterEnv{tabular}{%
  isLong=false, hasWidth=false, isTabu=false, hasCaption=false,
  packages={},
```

```
  atEnd={\preto\endtabular{#1}},
}
\metatblRegisterEnv{tabularx}{%
  isLong=false, hasWidth=true, isTabu=false, hasCaption=false,
  packages=tabularx,
  atEnd={%
```

Of the following two lines, the latter is for the case that the xltabular package is loaded, and the former is for the case that the package is not loaded.

```
    \preto\TX@endtabularx{\toks@\expandafter{\the\toks@#1}}%
    \preto\XLT@i@TX@endtabularx{\toks@\expandafter{\the\toks@#1}}},
}
\metatblRegisterEnv{longtable}{%
  isLong=true, hasWidth=false, isTabu=false, hasCaption=true,
  packages={longtable},
  atEnd={\preto\endlongtable{#1}},
}
\metatblRegisterEnv{xltabular}{%
  isLong=true, hasWidth=true, isTabu=false, hasCaption=true,
  packages=xltabular,
  atEnd={\preto\XLT@ii@TX@endtabularx{\toks@\expandafter{\the\toks@#1}}},
}
\metatblRegisterEnv{tabu}{%
  isLong=false, hasWidth=true, isTabu=true, hasCaption=false,
  packages={tabu},
```

The following is not a mistake: `tabu` does `\def\endtabu{\endtabular}` at the beginning of a `tabu` environment.

```
  atEnd={\preto\endtabular{#1}},
}
\metatblRegisterEnv{longtabu}{%
  isLong=true, hasWidth=true, isTabu=true, hasCaption=true,
  packages={tabu,longtable},
```

The following is not a mistake: `tabu` does `\def\endlongtabu{\endlongtable}` at the beginning of a `longtabu` environment.

```
  atEnd={\preto\endlongtable{#1}},
}
```

`\metatbl@ifhasXcolumns` The `\metatbl@ifhasXcolumns{`⟨*preamble*⟩`}{`⟨*iftrue*⟩`}{`⟨*iffalse*⟩`}` takes a ⟨*preamble*⟩ (the argument of a `tabular` environment that specifies the columns of the table) and checks, whether this preamble contains an "`X`" column. If such a column is contained, the macro expands to ⟨*iftrue*⟩. Otherwise, it expands to ⟨*iffalse*⟩.

```
\newrobustcmd\metatbl@ifhasXcolumns[1]{%
  \begingroup
```

The `\metatbl@@branch` macro is used at the end of the macro to select ⟨*iftrue*⟩ or ⟨*iffalse*⟩ for expansion. Initially, the macro is defined to select ⟨*iffalse*⟩.

```
  \def\metatbl@@branch{\@secondoftwo}%
```

The code uses the `\@mkpream` macro of the array package to create an `\halign` preamble from the `tabular` ⟨*preamble*⟩. The result of `\@mkpream` is in `\@preamble` afterwards, but this result is not used, but rather discarded at the `\endgroup`

below. Rather, we hook into \@mkpream via \NC@rewrite@X, which is used when an X column was encountered in ⟨*preamble*⟩.[2] When an X column is encountered, \metatbl@@branch is redefined to expand to ⟨*iftrue*⟩ in the end.

```
\def\NC@rewrite@X{\def\metatbl@@branch{\@firstoftwo}\NC@find}%
\@mkpream{#1}%
\expandafter\endgroup\metatbl@@branch}
```

10.7.3 Environment-Independent Parts

\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 (\begingroup and \endgroup) 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{%
  \kvt@setkeys{#3}{Row}%
  \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}}}%
  \ifbool{kvt@Row@uncounted}{}{%
    \appto\kvt@@row{\noalign{\kvt@stepcounters}}}%
```

[2] This hooking into \@mkpream is inspired by how tabularx replaces X columns by p columns as part of its measuring.

The following loop uses \do{⟨*cname*⟩} to append the content of all displayed 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}%
```

Apply expansion control options, but only to manually supplied cell values, not to default values.

```
     \ifbool{kvt@Row@expandonce}
       {\expandafter\let\expandafter\kvt@@cell\kvt@@cell}{}%
     \ifbool{kvt@Row@expand}
       {\protected@edef\kvt@@cell{\kvt@@cell}}{}}%
```

Separately also already create the formatted content.

```
  \ifcsvoid{kvt@@noformat@#2@##1}
    {\edef\kvt@@formatter{\expandonce{\csname kvt@col@format@#2@##1\endcsname}}}%
    {\def\kvt@@formatter{\@firstofone}}%
  \edef\kvt@@fmtcell{\expandonce\kvt@@formatter{%
    \expandonce\kvt@@cell}}%
  \csundef{kvt@@noformat@#2@##1}%
```

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
  \expandafter\appto\expandafter\kvt@@row\expandafter{\kvt@@fmtcell}%
}\dolistcsloop{kvt@displaycols@#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@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@CheckMulticolumn The \kvt@CheckMulticolumn{⟨*arg1*⟩}{⟨*arg2*⟩}{⟨*arg3*⟩}{⟨*arg4*⟩}\@undefined macro checks whether a cell's initial content (captured by ⟨*arg1*⟩ to ⟨*arg4*⟩, starts a multicolumn 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{%
  \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}}%
      {\expandonce\kvt@@formatter{\expandonce{#4}}}}%
  \fi}
```

10.7.4 Row Styles

\kvtNewRowStyle The \kvtNewRowStyle{⟨*name*⟩}{⟨*row-options*⟩} macro declares ⟨*name*⟩ as a row style and defines it to be equivalent to specifying ⟨*row-options*⟩ directly in the optional argument of \Row. The macro fails if ⟨*name*⟩ is already declared as a row style.

```
\newcommand\kvtNewRowStyle[2]{%
  \ifcsundef{kvt@@rowstyle@#1}
    {\csdef{kvt@@rowstyle@#1}{#2}}
    {\kvt@error{Row style '#1' is already defined}{Use
      \string\kvtRenewRowStyle\space to change an existing style.}}}
```

\kvtRenewRowStyle The \kvtRenewRowStyle{⟨*name*⟩}{⟨*row-options*⟩} macro re-defines an already existing row style with new ⟨*row-options*⟩.

```
\newcommand\kvtRenewRowStyle[2]{%
  \ifcsundef{kvt@@rowstyle@#1}
    {\kvt@error{Row style '#1' is not defined}
      {Use \string\kvtNewRowStyle\space to define a new row style.}}
    {\csdef{kvt@@rowstyle@#1}{#2}}}
```

\kvt@UseRowStyle The \kvt@UseRowStyle{⟨*style*⟩} macro sets the row keys based on the ⟨*row-options*⟩ stored for the given ⟨*style*⟩.

```
\newcommand\kvt@UseRowStyle[1]{%
  \ifcsundef{kvt@@rowstyle@#1}
    {\kvt@error{Row style '#1' is not defined}
      {Use \string\kvtNewRowStyle\space to define a new row style.}}
    {\kvt@setcskeys{kvt@@rowstyle@#1}{Row}}}
```

\kvt@UseRowStyles The \kvt@UseRowStyle{⟨*styles*⟩} macro sets the row keys based on the ⟨*row-options*⟩ for all styles in the comma-separated list ⟨*styles*⟩.

```
\newcommand\kvt@UseRowStyles[1]{%
```

We use \kvt@xkv@disablepreset to eliminate undesired effects from presets. When, for example, using \Row[bg=red,style=abc]{...}, this causes a \setkeys[kvt]{Row}{xyz} (if xyz is how the style abc is defined) inside the \setkeys[kvt]{Row}{bg=red,style=abc}. The former \setkeys would then again employ the presets for Row (e.g., from a \kvtSet{Row/bg=blue} and overwrite the bg=red.

```
  \kvt@xkv@disablepreset[kvt]{Row}{%
    \forcsvlist\kvt@UseRowStyle{#1}}}
```

\kvt@xkv@disablepreset \kvt@xkv@savepreset \kvt@xkv@restorepreset The \kvt@xkv@disablepreset[⟨*prefix*⟩]{⟨*family*⟩}{⟨*code*⟩} disables presets (head and tail) for ⟨*family*⟩ during the expansion of ⟨*code*⟩. The auxiliary macros \kvt@xkv@savepreset{⟨*prefix*⟩}{⟨*family*⟩}{⟨*h/t*⟩} and \kvt@xkv@restorepreset{⟨*prefix*⟩}{⟨*family*⟩}{⟨*h/* save+unset and, respectively, restore the preset keys for ⟨*family*⟩ – head keys for ⟨*h/t*⟩=h and tail keys otherwise.

```
\newcommand\kvt@xkv@disablepreset[3][KV]{%
  \ifnumgreater{\XKV@depth}{1}
    {#3}
    {\kvt@xkv@savepreset{#1}{#2}{h}%
     \kvt@xkv@savepreset{#1}{#2}{t}%
     #3%
     \kvt@xkv@restorepreset{#1}{#2}{h}%
     \kvt@xkv@restorepreset{#1}{#2}{t}}}
\newcommand\kvt@xkv@savepreset[3]{%
  \csletcs{kvt@@saved@preset#3}{XKV@#1@#2@preset#3}%
  \csundef{XKV@#1@#2@preset#3}}
\newcommand\kvt@xkv@restorepreset[3]{%
  \csletcs{XKV@#1@#2@preset#3}{kvt@@saved@preset#3}}
```

10.8 Collecting Key-Value Table Content

\NewCollectedTable The \NewCollectedTable{⟨*cname*⟩}{⟨*tname*⟩} macro registers a new table for recorded rows under name ⟨*cname*⟩ for table type ⟨*tname*⟩. The macro can only be used when ⟨*cname*⟩ is not already defined. It's function is not more than memorizing ⟨*tname*⟩ for ⟨*cname*⟩.

```
\newcommand\NewCollectedTable[2]{%
  \ifcsvoid{kvt@@tnameof@#1}
    {\csgdef{kvt@@tnameof@#1}{#2}}
    {\kvt@error{Name '#1' for a row collection is already defined}
      {Check for other \string\NewCollectedTable{#1}.}}}
```

\CollectRow The \CollectRow[⟨*options*⟩]{⟨*cname*⟩}{⟨*content*⟩} writes a \kvt@RecordedRow entry to the aux file. Fragile parts of ⟨*content*⟩ are protected through \protected@write.

```
\newcommand\CollectRow[3][]{%
  \ifcsvoid{kvt@@tnameof@#2}
    {\kvt@error{No row collection with name '#2' defined}
      {Use \string\NewCollectedTable in the preamble to define it.}}
    {%
```

First check in a local group whether the passed ⟨*content*⟩ and ⟨*options*⟩ are of a proper syntax.

```
      \begingroup
      \kvt@setkeys{#1}{Row}%
      \kvt@colsetcskeys{kvt@@tnameof@#2}{#3}%
      \endgroup
```

Next, write to \@auxout.

```
      \kvt@protected@write\@auxout{\string\kvt@RecordedRow{#1}{#2}{%
```

In the following, the columns' default values are explicitly added to the row. This ensures that defaults are expanded (via the \write) at the point at which a row is recorded rather than when the row is displayed. This allows using \thepage as the default value for a column with the intuitively expected outcome.

```
        \kvt@coldefaults{#2}%
        #3}}%
    }}
```

\kvt@protected@write The \kvt@protected@write{⟨*file*⟩}content macro writes ⟨*content*⟩ to ⟨*file*⟩. The write ensures that ⟨*content*⟩ is written in a particularly protected form that

1. protects ordinarily \protect'ed parts via \protected@write;

```
    \newcommand\kvt@protected@write[2]{\protected@write{#1}
```

2. protects table macros – like \thekvtRow –, which are stored in the etoolbox list \kvt@@writeprotected@cmds, by defining them to expand to their own name – delaying the actual expansion until when the file's contents is expanded;

```
        {\def\do##1{\def##1{\string##1}}%
         \dolistloop{\kvt@@writeprotected@cmds}%
```

3. protects table counters like kvtRow by adapting the counter-formatting macros to treat table counters differently from other counters.

```
         \forlistloop{\kvt@writeprotect@fmt}{\kvt@@numberformatters}}
        {#2}}
```

\kvt@writeprotect@fmt The \kvt@writeprotect@fmt{⟨*fmt-csname*⟩} macro takes the name of a counter-formatting macro (e.g., the name "arabic" for the macro\arabic) and redefines it such that counters declared via \kvtDeclareTableCounters are not expanded while all other counters are treated normally.

```
\newcommand\kvt@writeprotect@fmt[1]{%
```

First, save a copy of ⟨*fmt-csname*⟩ and then redefine ⟨*fmt-csname*⟩.

```
  \csletcs{kvt@@fmt@#1}{#1}%
  \csdef{#1}##1{%
```

The kvt@@c@##1 in the following condition is a csname that is defined by \kvtDeclareTableCounters if ##1 (the counter to be formatted) has been declared as a table counter. If the macro is defined, then ⟨*fmt-csname*⟩ expands to its name with its argument. Otherwise, the saved copy of ⟨*fmt-csname*⟩ is expanded, producing the actual counter value.

```
    \ifcsdef{kvt@@c@##1}
      {\expandafter\string\csname#1\endcsname{##1}}
      {\csname kvt@@fmt@#1\endcsname{##1}}}}
```

\kvtDeclareTableMacros The \kvtDeclareTableMacros{⟨*macro-list*⟩} macro declares all the macros in ⟨*macro-list*⟩ to be "table macros", i.e., macros that should be expanded inside the KeyValTable environment rather than in a \CollectRow. The macro records the ⟨*macro-list*⟩ by appending its elements to \kvt@@writeprotected@cmds. The actual expansion control is performed by \kvt@protected@write.

```
\newcommand\kvtDeclareTableMacros[1]{%
  \forcsvlist{\listadd\kvt@@writeprotected@cmds}{#1}}
```

\kvt@@writeprotected@cmds Initially empty etoolbox list of table macros.

```
\newcommand\kvt@@writeprotected@cmds{}
```

\kvtDeclareTableCounters The \kvtDeclareTableCounters{⟨*counter-list*⟩} macro declares all the counters in ⟨*counter-list*⟩ to be "table counters", i.e., counters that should be expanded inside the KeyValTable environment rather than in a \CollectRow. The macro only marks the counters by defining \kvt@@c@⟨*counter*⟩. The actual expansion control is performed by \kvt@writeprotect@fmt.

```
\newcommand\kvtDeclareTableCounters[1]{%
  \def\do##1{\cslet{kvt@@c@##1}\@ne}%
  \docsvlist{#1}}
```

\kvtDeclareCtrFormatters The \kvtDeclareCtrFormatters{⟨*macro-list*⟩} macro declares all the macros in ⟨*macro-list*⟩ to be counter-formatting macros, i.e., macros that take a LaTeX counter as their argument and format the counter's value, e.g., arabic, alphabetic, or as a roman number. The macro records the ⟨*macro-list*⟩ by appending the csnames of its elements to \kvt@@numberformatters. The actual expansion control for the macros in ⟨*macro-list*⟩ is performed by \kvt@writeprotect@fmt.

```
\newcommand\kvtDeclareCtrFormatters[1]{%
  \def\do##1{\listeadd\kvt@@numberformatters{%
    \expandafter\@gobble\string##1}}%
  \docsvlist{#1}}
```

\kvt@@writeprotected@cmds Initially empty etoolbox list of counter-formatting macros.

```
\newcommand\kvt@@numberformatters{}
```

The following registers the row counter macros as well as the row counters themselves as macros/counters that shall only be expanded inside the respective table.

```
\kvtDeclareTableMacros{\thekvtRow,\thekvtTypeRow,\thekvtTotalRow}
\kvtDeclareTableCounters{kvtRow,kvtTypeRow,kvtTotalRow}
```

The following registers macros that format counter values. This registering is necessary such that \kvt@writeprotect@fmt can protect table counters from expansion.

```
\kvtDeclareCtrFormatters{\arabic,\alph,\Alph,\roman,\Roman,\fnsymbol}
```

\kvt@coldefault
\kvt@coldefaults
\kvt@coldefaults@i The \kvt@coldefault{⟨*tname*⟩}{⟨*cname*⟩} macro expands to "⟨*cname*⟩={⟨*default*⟩},", where ⟨*default*⟩ is the default value of column ⟨*cname*⟩ in table type ⟨*tname*⟩. If ⟨*default*⟩ is empty, then the macro expands to the empty string. The \kvt@coldefaults@i{⟨*tname*⟩} macro expands to the comma-separated list of the \kvt@coldefault for all *displayed* columns of table type ⟨*tname*⟩. Finally, the

\kvt@coldefaults{⟨*cname*⟩} macro expands to \kvt@coldefaults for the table type assigned to ⟨*cname*⟩ via \NewCollectedTable.

```
\newcommand\kvt@coldefaults[1]{%
  \kvt@coldefaults@i{\csuse{kvt@@tnameof@#1}}}
\newcommand\kvt@coldefaults@i[1]{%
  \forlistcsloop{\kvt@coldefault{#1}}{kvt@displaycols@#1}}
\newcommand\kvt@coldefault[2]{\ifcsvoid{kvt@col@default@#1@#2}{}{%
  #2={\csuse{kvt@col@default@#1@#2}},}}
```

\kvt@RecordedRow The \kvt@RecordedRow{⟨*options*⟩}{⟨*cname*⟩}{⟨*content*⟩} appends a \Row with ⟨*options*⟩ and ⟨*content*⟩ to a global macro for ⟨*cname*⟩.

```
\newcommand\kvt@RecordedRow[3]{%
  \csgappto{kvt@@rowsof@#2}{\Row[{#1}]{#3}}}
```

\ShowCollectedTable The \ShowCollectedTable[⟨*options*⟩]{⟨*cname*⟩} produces a KeyValTable table for the rows stored under the given ⟨*cname*⟩, table options ⟨*options*⟩.

```
\newcommand\ShowCollectedTable[2][]{%
  \ifcsvoid{kvt@@tnameof@#2}
    {\kvt@error{No row collection with name '#2' defined}
      {Use \string\NewCollectedTable in the preamble to define it.}}
    {\ifcsvoid{kvt@@rowsof@#2}
      {\kvt@warn{No row data available for name '#2'.
        A LaTeX rerun might be needed^^M
        for the row data to be available}%
       \kvt@tableofcname{#2}{#1}{???\tabularnewline}}%
      {\kvt@tableofcname{#2}{#1}{\csuse{kvt@@rowsof@#2}}}}}
```

\kvt@tableof \kvt@tableofcname \kvt@tableofcname@i The \kvt@tableof{⟨*tname*⟩}{⟨*options*⟩}{⟨*content*⟩} expands to a KeyValTable environment for table type ⟨*tname*⟩ with ⟨*options*⟩ and environment body ⟨*content*⟩. The \kvt@tableofcname{⟨*cname*⟩}{⟨*options*⟩}{⟨*content*⟩} expands to a \kvt@tableof where ⟨*tname*⟩ is the table type assigned to ⟨*cname*⟩. Finally, \kvt@tableofcname@i is an auxiliary macro for expansion control.

```
\newcommand\kvt@tableof[3]{%
  \begin{KeyValTable}[{#2}]{#1}%
    #3%
  \end{KeyValTable}}
\newcommand\kvt@tableofcname[1]{\expandafter
  \kvt@tableofcname@i\expandafter{\csname kvt@@tnameof@#1\endcsname}}
\newcommand\kvt@tableofcname@i[1]{\expandafter
  \kvt@tableof\expandafter{#1}}
```

10.8.1 Table Content from Files

\ShowKeyValTableFile The \ShowKeyValTableFile[⟨*options*⟩]{⟨*tname*⟩}{⟨*filename*⟩} macro typesets a KeyValTable environment of type ⟨*tname*⟩ with the given ⟨*options*⟩. The body of the environment (i.e., the rows of the table) are read from the file ⟨*filename*⟩.

```
\newcommand\ShowKeyValTableFile[3][]{%
  \IfFileExists{#3}
    {\begin{KeyValTable}[{#1}]{#2}\@@input#3 \end{KeyValTable}}%
    {\kvt@error{No KeyValTable file '#3'}
```

```
      {Check whether the file really exists or whether there is a
       typo in the argument '#3'}}}
```

10.8.2 Legacy Variant

\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
    {\begingroup}
    {\csxappto{kvt@rows@#1}{\expandonce{\kvt@@row}}\endgroup}
    {#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}}}{}%
```

10.9 Package Options

The following option allows specifying a version for (hopefully) compatibility with the respective old version.

```
\define@cmdkey[kvt]{PackageOptions}[kvt@@pkg@]{compat}{}
```

Next, set default package options and process them.

```
\ExecuteOptionsX[kvt]<PackageOptions>{%
  compat=2.0,
}
\ProcessOptionsX[kvt]<PackageOptions>\relax
```

10.10 Compatibility

\kvt@NewCompat The \kvt@IfVersion{⟨*relation*⟩}{⟨*version*⟩}{⟨*iftrue*⟩}{⟨*iffalse*⟩} macro expands to ⟨*iftrue*⟩ if the requested package version is in the given ⟨*relation*⟩ (<, <, or =)

to ⟨*version*⟩. Otherwise, the macro expands to ⟨*iffalse*⟩. Package versions are requested via the `compat` package option. If no version is explicitly requested, the newest version is implicitly assumed to be requested. ⟨*code*⟩ as

```
\newcommand\kvt@IfVersion[2]{%
  \ifdimcomp{\kvt@@pkg@compat pt}{#1}{#2pt}}
```

Before v2.0, tabu was the default table environment.

```
\kvt@IfVersion{<}{2.0}{%
  \metatblRequire{tabu,longtabu}
  \kvt@DefineStdTabEnv[onepage]{tabu}
  \kvt@DefineStdTabEnv[multipage]{longtabu}
}{%
  \metatblRequire{tabularx,longtable,xltabular}
  \kvt@DefineDualTabEnv{onepage}{tabular}{tabularx}
  \kvt@DefineDualTabEnv{multipage}{longtable}{xltabular}
}
```

Before v2.0, the second optional argument of `\NewKeyValTable` specified the header rows only. Only afterwards, that argument received a key-value syntax.

```
\kvt@IfVersion{<}{2.0}{%
  \let\kvt@parselayout=\kvt@parseheadrows
}{}
```

Change History

v0.1
- General: Initial version 1

v0.2
- `\NewKeyValTable`: Added table-type options 26
- `\kvtLabel`: Added macro for row labeling 35
- General: Added “shape” table option 24

v0.3
- `\kvt@StartTabularlike`: Added `showhead` option 36
- `\kvtLabel`: Robustified for use with, e.g., cleveref 35
- `\kvtStrutted`: Fix for cells with vertical material 23

v0.3b
- General: Package author’s name change 1

v1.0
- `\NewKeyValTable`: Added optional headers argument 26
- Added zero-width column for `\multicolumn` 27
- `\kvt@AddKeyValRow`: Added [⟨*options*⟩] 41
- `\kvt@AddKeyValRow@ii`: Added `\multicolumn` support 42
- `\kvt@StartTabularlike`: Added `width` option 36
- Implemented `showrules` option 36
- General: Enabled default “true” for “hidden” 24

v2.0
- `\CollectRow`: Added the macro . . 44
- `\NewCollectedTable`: Added the macro 44
- `\NewKeyValTable`: Changed headers argument to layout argument 26
- `\ShowCollectedTable`: Added the macro 47
- `\ShowKeyValTableFile`: Added the macro 47
- `\kvtNewRowStyle`: Added the macro 43
- `\kvtRenewRowStyle`: Added the macro 43

\kvtStrutted: Added optional argument 23
General: added package option “compat” 48
added row option “style” 25
added row option “uncounted” 25
added row options “expand” and “expandonce” 25
added row options “nobg” and “norowbg” 25
added table options “caption” and “label” 24

Index

Symbols

\@@input 655
\@auxout 598
\@dblarg 400
\@empty 280, 303, 305, 503
\@firstofone 22, 99, 526
\@firstoftwo 184, 492
\@gobble 621
\@ifnextchar 121, 497
\@mkpream 493
\@ne 285, 292, 313, 617
\@nil 142, 183, 393, 394
\@secondoftwo 185, 491
\@undefined . 139, 145, 196, 199, 278, 314, 535, 547
\\ 9

A

above (option-key) *11*
\AddKeyValRow *5*, 664, 670
\addtocounter . 334, 544, 545, 546
\advance 285, 531, 549
\AfterEndEnvironment 347
align (option-key) *3*, *14*, *15*
\Alph 626
\alph 626
\appto 29, 267, 270, 271, 295, 306, 359, 362, 365, 514, 533, 536, 538
\arabic 626
around (option-key) *11*
\arrayrulewidth 187

B

\begin 646, 655, 660
\begingroup . 194, 262, 277, 350, 371, 490, 594, 666
below (option-key) *11*
bg (option-key) *11*
\bgroup 342

C

\caption 363
caption (option-key) *16*
\cmdkvt@Row@above 508, 510
\cmdkvt@Row@below 539, 541
\cmdkvt@Row@bg 511, 512
\cmdkvt@Table@caption . 360, 363
\cmdkvt@Table@headalign . . 177, 179
\cmdkvt@Table@headbg . . 172, 297
\cmdkvt@Table@headformat . 178, 180
\cmdkvt@Table@label . . 364, 366
\cmdkvt@Table@rowbg . . 376, 389
\cmdkvt@Table@shape . . 344, 346
\cmdkvt@Table@width . . 380, 381
\CollectRow *5*, 589
counters:
kvtRow *7*
kvtTotalRow *7*
kvtTypeRow *7*
\csappto 142, 161, 296
\csdef . . 51, 60, 66, 68, 70, 77, 79, 129, 130, 132, 133, 134, 135, 136, 141, 167, 168, 234, 260, 261, 324, 348, 555, 562, 609, 663
\cseappto 159, 162, 268
\csedef . . 131, 271, 404, 409, 412
\csexpandonce 159, 162, 218, 219, 269, 311, 312, 351, 383
\csgappto 634
\csgdef 586
\cslet 313, 617
\csletcs 275, 580, 583, 608
\csname . 16, 21, 34, 256, 377, 402, 405, 407, 410, 413, 418, 438,

453, 456, 525, 611, 612, 650
\csundef 529, 581
\csuse 272, 338, 344, 346, 357, 370, 374, 386, 628, 632, 644, 661
\csxappto 667

D

\DeclareListParser 7, 8, 9
default (option-key) 3
\define@boolkey . . 37, 44, 45, 56, 82, 88, 89, 90, 425, 426, 427, 429
\define@choicekey 49
\define@cmdkey 35, 36, 41, 42, 43, 46, 47, 65, 67, 69, 72, 74, 81, 83, 84, 165, 213, 431, 671
\define@key 39, 52, 53, 54, 55, 62, 76, 78, 85, 87, 166, 432
\do 138, 196, 264, 278, 283, 296, 299, 447, 450, 516, 603, 617, 620
\docsvlist . . . 448, 451, 618, 622
\dolistcsloop 293, 537
\dolistloop 604

E

\eappto . . 212, 308, 509, 512, 540
\egroup 342
\else 184
\end 648, 655, 662
\endcsname . 16, 21, 34, 256, 377, 402, 405, 407, 410, 413, 418, 438, 453, 456, 525, 611, 612, 650
\endgroup 198, 274, 299, 350, 371, 494, 597, 667
\endhead 390
\endlongtable 472, 487
\endtabular 460, 482
environments:
KeyValTable 4, 340
KeyValTableContent . 6, 669
\ExecuteOptionsX 672
expand (option-key) 19
\expandafter 14, 16, 19, 21, 23, 73, 75, 148, 184, 185, 198, 241, 243, 254, 256, 266, 274, 286, 299, 342, 365, 366, 377, 402, 405, 407, 410, 413, 416, 417, 437, 455, 456, 466, 467, 477, 494, 521, 534, 536, 611, 621, 649, 650, 651, 652
\expandonce . 131, 178, 179, 180, 216, 217, 298, 310, 376, 380, 381, 389, 510, 512, 525, 527, 528, 541, 551, 667
expandonce (option-key) 19

F

\fi . . 23, 185, 242, 250, 286, 552
\fnsymbol 626
\forcsvlist 64, 570, 614
\forlistcsloop . . . 155, 252, 630
\forlistloop 605
format (option-key) 3, 15

H

head (option-key) 3, 15
headalign (option-key) 9
headbg (option-key) 9
headformat (option-key) 9
hidden (option-key) 3, 11
\hspace 187

I

\ifbool . . 356, 373, 385, 442, 443, 444, 445, 502, 513, 520, 522
\ifcase 241, 243
\ifcsdef 287, 610
\ifcsmacro 320
\ifcsstring 158
\ifcsundef 554, 559, 564
\ifcsvoid 160, 230, 240, 517, 524, 585, 590, 631, 636, 639
\ifdefempty 360, 364
\ifdefequal 284, 303
\ifdefvoid . 26, 31, 177, 214, 287, 307, 508, 511, 539
\ifdimcomp 677
\IfFileExists 654
\ifhmode 23
\ifinlist 125
\ifinlistcs 152, 200, 205, 225, 316
\ifnum 286
\ifnumgreater 530, 572
\ifnumodd 395
\ifstrempty . 258, 300, 336, 337, 393, 399, 532

\ifstrequal 266
\ifx 183, 548

K

KeyValTable (environment) 4, 340
KeyValTableContent (environment) 6, 669
\kvt@@cell 518, 519, 521, 523, 528, 532, 534
\kvt@@coldo 239, 252
\kvt@@colgrp 66, 68, 70, 209, 211, 246
\kvt@@colgrp@first 214, 216, 237, 244
\kvt@@colgrp@n . . . 217, 236, 251
\kvt@@colreg 315, 325
\kvt@@column 59, 60, 146, 147, 148, 151, 170
\kvt@@curgrp 248
\kvt@@curhd . 282, 283, 284, 287, 288, 290, 292
\kvt@@do 350, 352, 371, 391
\kvt@@endhook . 355, 359, 362, 365
\kvt@@extraalign . 304, 305, 310
\kvt@@fmtcell . 527, 532, 536, 550
\kvt@@formatter 525, 526, 527, 551
\kvt@@hdcell 77, 79, 326
\kvt@@lasthd . 282, 284, 292, 307, 311, 312, 313
\kvt@@numberformatters 605, 620, 623
\kvt@@parseheadrows . 263, 267, 270, 271, 274
\kvt@@pkg@compat 677
\kvt@@presetqueue . . . 26, 27, 29
\kvt@@psvdo 224, 235
\kvt@@recenttable 348, 354
\kvt@@result 195, 198, 212
\kvt@@row 342, 503, 507, 509, 512, 514, 533, 536, 538, 540, 667
\kvt@@rule 357, 358, 359, 384, 386
\kvt@@span 281, 285, 286, 292, 301, 309, 515, 530, 531, 549
\kvt@@status . 238, 241, 242, 243, 244
\kvt@@tmp 265, 266
\kvt@@tmpgrphd 280, 295, 298, 303, 306, 308
\kvt@@tname . . . 73, 75, 190, 192
\kvt@@writeprotected@cmds 604, 614, 615, 623
\kvt@AddKeyValRow 341, 495, 665
\kvt@AddKeyValRow@i . 498, 499, 500
\kvt@AddKeyValRow@ii . . 504, 505
\kvt@alltables . . . 125, 137, 188
\kvt@checkcolgroup . . . 223, 255
\kvt@checkcolgroupcs . . 211, 253
\kvt@CheckMulticolumn . 534, 547
\kvt@coldefault 627
\kvt@coldefaults 599, 627
\kvt@coldefaults@i 627
\kvt@colkeysetter 52, 53, 54, 55, 57, 58
\kvt@colsetcmdkeys 17
\kvt@colsetcskeys 17, 596
\kvt@colsetkeys 17
\kvt@concludecolumn . 286, 294, 302
\kvt@def@globalopt 61
\kvt@def@globalopts 61, 71, 80, 91
\kvt@defaultheader . . . 136, 171
\kvt@defaultheader@i . 173, 174, 181
\kvt@DefineDualTabEnv 406, 684, 685
\kvt@DefineStdTabEnv . 400, 419, 420, 421, 422, 423, 424, 680, 681
\kvt@DefineStdTabEnv@i 400, 401
\kvt@dobrklist 9, 273
\kvt@dossvlist . 7, 140, 197, 279
\kvt@dottedrowcolors . . 372, 392
\kvt@dottedrowcolors@i 393, 394
\kvt@error 10, 126, 153, 200, 205, 225, 230, 246, 288, 317, 321, 556, 560, 565, 587, 591, 637, 656
\kvt@forpsvlist 8, 235, 325
\kvt@HackIntercolSpace 131, 186, 304
\kvt@ifhasXcolumns . . . 408, 415
\kvt@ifnil 175, 182
\kvt@IfVersion . . . 676, 678, 687
\kvt@keysetter 30, 59

\kvt@LabelCtr 339
\kvt@lazypreset 28, 32, 62
\kvt@NewCompat 676
\kvt@NewKeyValTable . 122, 123, 124
\kvt@parsecolgroup . . . 196, 199
\kvt@parsecolgroups . . 75, 193
\kvt@parsecolspec 139, 145
\kvt@parsecolspec@i . . 148, 149
\kvt@parsecolspec@ii . . 149, 150
\kvt@parsehdcolspec . . 278, 314
\kvt@parseheadrow 270, 276
\kvt@parseheadrows 73, 257, 688
\kvt@parseheadrows@i . . 258, 259
\kvt@parselayout . 143, 189, 688
\kvt@protected@write . . 598, 602
\kvt@RecordedRow 598, 633
\kvt@rowcolorornot 172, 297, 300
\kvt@setcmdkeys 12
\kvt@setcskeys 12, 567
\kvt@setkeys . . 12, 25, 38, 40, 86, 157, 191, 210, 327, 351, 501, 595
\kvt@SetOptions 343, 349
\kvt@StartTabularlike 353, 403, 411, 414
\kvt@stepcounters 514, 543
\kvt@tableof 645
\kvt@tableofcname 643, 644, 645
\kvt@tableofcname@i 645
\kvt@taburowcolors . . . 389, 398
\kvt@UseRowStyle 563, 570
\kvt@UseRowStyles 87, 568
\kvt@warn 10, 367, 640
\kvt@writeprotect@fmt . 605, 607
\kvt@xkv@disablepreset 569, 571
\kvt@xkv@restorepreset . . . 571
\kvt@xkv@savepreset 571
\kvtDeclareCtrFormatters *8*, 619, 626
\kvtDeclareTableCounters *8*, 616, 625
\kvtDeclareTableMacros . *8*, 613, 624
\kvtLabel *7*, 332
\kvtNewRowStyle *12*, 553, 561, 566
\kvtRenewRowStyle . *12*, 557, 558
\kvtRow 328
kvtRow (counter) *7*
\kvtSet *13*, 24, 94
\kvtStrutted *11*, 22, 105, 117
\kvtTableOpt 34
\kvtTotalRow 330
kvtTotalRow (counter) *7*
\kvtTypeRow 329
kvtTypeRow (counter) *7*

L

\label 337, 366
label (option-key) *16*
\let . 237, 280, 292, 304, 305, 355, 503, 521, 688
\letcs 283, 518, 519
\linewidth 102
\listadd 137, 614
\listcsadd 156, 163, 222
\listeadd 620

M

\m@ne 531, 549
\metatbl@@branch . 491, 492, 494
\metatbl@@envname 434, 438, 440, 441
\metatbl@boolprop 425, 426, 428, 430, 439
\metatbl@csnamearg 447, 450, 454
\metatbl@ifhasXcolumns 416, 489
\metatbl@setprop . 431, 432, 436
\metatblAtEnd 355, 452
\metatblHasCaption . . . 361, 442
\metatblHasWidth 378, 442
\metatblIsLong 390, 442
\metatblIsTabu 372, 379, 388, 442
\metatblRegisterEnv . 433, 457, 462, 469, 474, 479, 484
\metatblRequire . . 446, 679, 683
\metatblUsePackage 446
\multicolumn . 179, 216, 308, 548, 550

N

\NC@find 492
\NC@rewrite@X 492
\NewCollectedTable . *4*, 584, 592, 638
\newcount 301
\newcounter . . 328, 329, 330, 339

\NewKeyValTable 2, 120, 203, 207, 228, 249, 291, 318, 322
\newrobustcmd 433, 489
\noalign 342, 509, 514, 540
nobg (option-key) 10
\noexpand 172, 175, 179, 213, 215, 216, 222, 296, 297, 298, 308, 350, 372, 377, 384, 386, 389, 390, 405, 410, 413, 509, 512, 540
norowbg (option-key) 10
\numexpr 251, 272, 374

O

option-keys:
above 11
align 3, 14, 15
around 11
below 11
bg 11
caption 16
default 3
expand 19
expandonce 19
format 3, 15
head 3, 15
headalign 9
headbg 9
headformat 9
hidden 3, 11
label 16
nobg 10
norowbg 10
rowbg 10
shape 9
showhead 9
showrules 9
span 15
style 12
uncounted 7
width 9
\or 241, 245

P

\PackageError 10
\PackageWarning 11
\PassOptionsToPackage 4
\presetkeys 29, 169
\preto 460, 466, 467, 472, 477, 482, 487
\ProcessOptionsX 675
\protected@edef 523
\protected@write 602
\providebool 440

R

\refstepcounter 335
\relax 183, 237, 251, 272, 355, 374, 515, 535, 549, 675
\RequirePackage 1, 2, 3, 5, 6, 450
\Roman 626
\roman 626
\Row 4, 341, 634, 670
rowbg (option-key) 10
\rowcolor 300, 362, 512
\rowcolors 396, 397

S

\setbool 441
\setcounter . . 331, 333, 369, 370
\setkeys . . . 12, 17, 215, 435, 506
shape (option-key) 9
\ShowCollectedTable . . . 5, 635
showhead (option-key) 9
\ShowKeyValTable 6, 659
\ShowKeyValTableFile . . . 5, 653
showrules (option-key) 9
\space 557, 561, 566
span (option-key) 15
\string . . 128, 203, 207, 228, 248, 249, 290, 291, 318, 322, 557, 561, 566, 588, 592, 598, 603, 611, 621, 638
\strut 23
style (option-key) 12

T

\tabularnewline 175, 295, 538, 643
\taburowcolors 399
\the . 251, 272, 309, 374, 466, 467, 477
\thekvtRow 624
\thekvtTotalRow 624
\thekvtTypeRow 348, 624
\toks@ 466, 467, 477
\trim@post@space@in 265

\trim@spaces@in 147
\TX@endtabularx 466

U

uncounted (option-key) 7
\undef 27, 170, 192, 282
\unexpanded . 176, 178, 180, 219, 297, 298, 351, 550
\usepackage 447

V

\value 333
\vspace 509, 540

W

width (option-key) 9

X

\XKV@depth 572
\XLT@i@TX@endtabularx 467
\XLT@ii@TX@endtabularx . . . 477

Z

\z@ 281, 286