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 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 6 Customizing the Layout 15
2 Defining Table Types 2 7 Use with Other Packages 22
3 Typesetting Tables 3 8 Related Packages 25
4 Row Numbering & Labeling 6 9 Future Work 25
5 Changing the Appearance 9 10 Implementation 26

∗This document corresponds to keyvaltable v2.2, dated 2020/04/05. 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.

```latex
\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}
```

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.

```latex
\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}
```

2 Defining Table Types

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

```
\NewKeyValTable{\langle tname \rangle}{\langle cols \rangle}{\langle layout \rangle}
```

The macro defines a table type with name `\langle tname \rangle` whose columns are specified by `\langle cols \rangle`. The `\langle layout \rangle` parameter must be a semicolon-separated list. Each column specification is of the form
In such a specification, \langle colname \rangle represents the name of the column. The \langle property \rangle=\langle value \rangle pairs configure certain properties of the column. The \langle property \rangle can be one of the following:

- \textbf{align=\langle value \rangle} \quad \textit{initially: } \text{l}
  This property specifies the alignment of content in the column. The \langle value \rangle can be set to any column alignment understood by table environments.\footnote{More complex values, for instance using the notation of the \texttt{array} package for inserting material before or after a column, are permitted but not further tested. Use at your own risk.}

- \textbf{default=\langle content \rangle} \quad \textit{initially: } \text{(empty)}
  This property specifies the default \langle content \rangle of a cell in this column, i.e., in case that a \texttt{\Row} does not provide content for the cell. Initially (i.e., if unset for a column), this is an empty string.

- \textbf{format=\langle single argument macro \rangle} \quad \textit{initially: } \texttt{\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 \texttt{\strut} before and after the content (to yield a better vertical row spacing).

- \textbf{head=\langle content \rangle} \quad \textit{initially: } \text{\langle colname \rangle}
  This property specifies the \langle content \rangle of the column’s header row. The initial value for this property is the name of the column.

- \textbf{hidden=true, false} \quad \textit{default: } true, \textit{initially: false}
  This property specifies whether a table column shall be displayed or not. The \langle value \rangle for this property can be true (to hide the cell) or false (to display the cell). Using hidden without \langle value \rangle is equivalent to specifying hidden=true.

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

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

\begin{table}[h]
\centering
\begin{tabular}{ll} 
\hline
\texttt{article} & \texttt{amount} \\
melon & 1 \\
apples & 6 \\
bicycle & 1 \\
\hline
\end{tabular}
\end{table}

The \langle options \rangle and \langle layout \rangle parameters of \texttt{\NewKeyValTable} are described in Section 5.1 and, respectively, Section 6.1 of this documentation.

3 Typesetting Tables

The \texttt{keyvaltable} package offers three possibilities for typesetting tables. The first is in the traditional \LaTeX \texttt{format}, 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 \( \langle cname \rangle \) for a new collection of rows. The collection is associated with the table type \( \langle tname \rangle \). This macro must be used before \CollectRow for a \( \langle cname \rangle \).

\CollectRow[{\langle options\rangle}]{\langle cname\rangle}{\langle content\rangle}

This macro adds the row content \( \langle content \rangle \) and row options \( \langle options \rangle \) to the row collection \( \langle cname \rangle \).

\ShowCollectedTable[{\langle options\rangle}]{\langle cname\rangle}

This macro typesets a table of the row collection \( \langle cname \rangle \), with the table options \( \langle options \rangle \). 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[{\langle options\rangle}]{\langle tname\rangle}{\langle filename\rangle}

This macro produces a KeyValTable environment of type \( \langle tname \rangle \) whose content is taken from the file \( \langle filename \rangle \). The \( \langle options \rangle \) 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}
```

### 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{\langle tname \rangle}{\langle options \rangle}{\langle content \rangle}
A table row is produced by the AddKeyValRow macro. The \textit{\langle tname \rangle} identifies the table type and the \textit{\langle content \rangle} provides the content of the cells in the row. The format of the \textit{\langle content \rangle} is the same as for the Row macro described in Section 3.

\ShowKeyValTable[\textit{\langle options \rangle}]{\textit{\langle tname \rangle}}

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}{\textit{\langle tname \rangle}}\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}

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=\the\kvtRow; text: align=l, head=Text}\begin{KeyValTable}{Numbered}\Row{text=First row}\Row{text=Second row}\end{KeyValTable}
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.

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

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.

By default, all counters start at value 1. Through the following possibilities, this behavior can be changed.

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.

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

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.

```latex
\begin{KeyValTable}{Labeled}
\Row{label: align=r, head=\textbf{\#}, text: align=l, head=\textbf{Text}}
\Row{\kvtLabel\{kvtRow\};}
\Row{\kvtLabel\{\ref{first}\}}
\Row{\kvtLabel\{}\langle\kvtRow\rangle\langle\text{label}\rangle\}
\end{KeyValTable}
```

The `\kvtLabel` macro shows the current value of the `\langle\kvtRow\rangle` – in particular `kvtRow`, `kvtTypeRow`, and `kvtTotalRow` – and sets the `\langle\text{label}\rangle` to the value of `\langle\kvtRow\rangle`. When using the macro with the `format` property, only the first argument `\langle\kvtRow\rangle` must be provided, as the above example shows. The second argument `\langle\text{label}\rangle` 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
```

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}}}
{\kvtDeclareCtrFormatters{{\macro-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.

```
{\kvtDeclareTableMacros{{\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-formattong 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 \langle options\rangle 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, \langle options\rangle must be specified as a comma-separated list of \langle property\rangle=\langle value\rangle pairs. The following \langle property\rangle keys can be configured.

backend = tabular, tabularx, longtable, xltabular, tabu, longtabu
shape = multipage, onepage

The backend property specifies the table environment to be used for producing the table. A set of six environments is currently supported, including environments that can span multiple pages and environments whose columns can stretch/shrink to fill the available space ("X" columns). The shape property abstracts from the concrete environments. In case of multipage, the table may span multiple pages and depending on whether X-columns are used or not, an appropriate environment is selected. In case of onepage, the table does not split into multiple pages. See Section 6.4 for more details on the available shapes and backends. Only one of shape and backend can be specified. If both are specified, the property that is specified last wins.

width = \langle dimension\rangle

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

valign = t, c, b
halign = l, c, r

These two properties specify the vertical and, respectively, horizontal alignment of the table, if the selected shape/backend supports it (see Section 6.4).

showhead = true, false

This property specifies whether the header row shall be shown. The \langle value\rangle 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
norules = true, false

The showrules property specifies whether top and bottom rules as well as a rule below the header row are drawn (true) or not (false). The norules property
serves the same purpose, but the value `true` hides the rules and the value `false` causes the rules to be drawn. Note that both properties only affect the rules that \keyvaltable produces automatically; rules manually added, e.g., via `\hline` or `\midrule` are not affected by the properties.

\begin{itemize}
  \item \texttt{headalign} = \langle \text{empty} \rangle \text{ or } \langle \text{coltype} \rangle \quad \text{initially:} \quad \langle \text{empty} \rangle
  
  This property specifies the alignment for header cells. If left empty, each header cell receives the same alignment as the respective column.

  \item \texttt{headbg} = \langle \text{color} \rangle \quad \text{initially:} \quad \text{black!14}
  
  This property specifies the background color of the header rows. The \langle \text{color} \rangle must be a single color specification that is understood by the \texttt{xcolor} package. The \langle \text{color} \rangle is passed directly to the `\rowcolor` macro. If \langle \text{color} \rangle is empty, then no background color is produced for the header row.

  \item \texttt{headformat} = \langle \text{single argument macro} \rangle \quad \text{initially:} \quad \langle \text{"identity"} \rangle
  
  This property specifies a format to be applied to all header cells. The value specified for the \texttt{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.

  \item \texttt{rowbg} = \langle \text{color} \rangle \quad \text{initially:} \quad \text{white..black!10}
  
  This property specifies the background colors of content rows. The \langle \text{value} \rangle for this property must be of the format \langle \text{oddcolor} \rangle..\langle \text{evencolor} \rangle. The first row after the header is colored with \langle \text{oddcolor} \rangle, the second row with \langle \text{evencolor} \rangle, and so forth. Both colors must be understood by the \texttt{xcolor} package. If \langle \text{color} \rangle is empty, then no background color is produced for content rows.

  \item \texttt{norowbg} = \text{true, false} \quad \text{default:} \quad \text{true, initially: false}
  \texttt{nobg} = \text{true, false} \quad \text{default:} \quad \text{true, initially: false}
  
  These properties are shorthands for \texttt{rowbg=\{\}} (turning off background colors for content rows) and, respectively, for \texttt{rowbg=\{\}},\texttt{headbg=\{\}} (turning off background colors for header rows and for content rows). Using these options without a value is equivalent to using \texttt{true} for the value. For instance, \texttt{nobg} is equivalent to \texttt{nobg=true}.

  Figure 1 on the following page demonstrates the \langle \text{options} \rangle in examples.
\end{itemize}

5.1.1 Table Styles and Resumable Options

Rather than specifying properties for individual tables or table types, \keyvaltable also supports named table styles.

\begin{itemize}
  \item \texttt{style} = \langle \text{list of style names} \rangle \quad \text{initially:} \quad \langle \text{empty} \rangle
  
  Through this property of tables or table types, a list of styles can be applied to a single table or, respectively, a table type. Each style must have been defined with `\kvtNewTableStyle` before.

\end{itemize}
\begin{KeyValTable}{TabOptions}
\Row{opt=showhead, val=false}
\Row{opt=rowbg, val=blue!10..blue!15}
\end{KeyValTable}

\begin{KeyValTable}{TabOptions2}
\Row{opt=showrules, val=false}
\Row{opt=headbg, val=blue!25}
\Row{opt=headalign, val=c}
\Row{opt=headformat, val=\textbf}
\Row{opt=norowbg, val=true}
\Row{opt=halign, val=r}
\end{KeyValTable}

\begin{KeyValTable}{TabOptions3}
\Row{opt=nobg}
\Row{opt=norules}
\end{KeyValTable}

\begin{KeyValTable}{TabOptions3}
\Row{opt=shape=onepage}
\Row{opt=valign=t}
\Row{opt=width=3cm}
\end{KeyValTable}

\begin{KeyValTable}{TabOptions2}
\Row{opt=showrules, val=false}
\Row{opt=headbg, val=blue!25}
\Row{opt=headalign, val=c}
\Row{opt=headformat, val=\textbf}
\Row{opt=norowbg, val=true}
\Row{opt=halign, val=r}
\end{KeyValTable}

\begin{KeyValTable}{TabOptions3}
\Row{opt=nobg}
\Row{opt=norules}
\end{KeyValTable}

\begin{KeyValTable}{TabOptions3}
\Row{opt=shape=onepage}
\Row{opt=valign=t}
\Row{opt=width=3cm}
\end{KeyValTable}

Figure 1: Examples for table options
This macro declares a new table style with the given \langle name \rangle and defines it to be equivalent to using the given \langle options \rangle. The \langle name \rangle must not already be defined.

\kvtRenewTableStyle\{(name)\}{(options)}

This macro re-defines an existing table style \langle name \rangle with new \langle options \rangle.

The following example demonstrates table styles for an individual table.

\kvtNewTableStyle\{plain\}{norules,nobg,headformat=\textbf}
\begin{KeyValTable}\[style=plain]\{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}

The \langle options \rangle in \kvtNewTableStyle can be left empty. In this case, the table style does not have any effect on the appearance of tables. However, the style can already be used for “tagging” tables and table types, while the final options for the style can be configured at a later point in time.

Even without table styles, the appearance of the previous KeyValTable can be used again through the following option.

resume* = true, false  \hspace{1cm} default: true, initially: false

When set to true, this option makes the table use the options from the previous KeyValTable environment. This option also implies the resume option (see Section 4.1 on page 7).

If the previous environment also used resume*, then the options of its predecessor environment are used, and so forth. Note that this means that table options are not accumulated over subsequent uses of resume*. This behavior is the same as in the enumitem package.

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 \langle arg \rangle and a \strut after \langle arg \rangle. This has the effect that the first and last row of \langle arg \rangle 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 \langle options \rangle 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.
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.

This property specifies the alignment of the cells in the row. If this property is not specified, the respective columns’ alignment is used. The alignment applies to normal cells as well as to cells in column groups.  

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.

These properties specify formatting for all cells of the particular row. The difference between the three properties is how they interact with the column formats of the respective cells in the row. The format property is applied to the cell content before the column format, and the format* property is applied after the column format. The format! property overrides any column formats in the respective row and also renders the format and format* properties ineffective.

This property, when used without a value or with value true, specifies that the row shall be formatted like a header row. Concretely, the alignment, background color, and format of the row’s cells is then set to the values of the table’s headalign, headbg, and headformat properties.

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/\langle option\rangle=\langle value\rangle} (see also Section 5.5).

The following example demonstrates some of 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, format=\textit\] {step=serve with a smile}  
\end{KeyValTable}  

Note that the alignment does not override the alignment specified in any \multicolumn if it is assigned to a cell in the row.
5.3.1 Row Styles

Rather than specifying properties for individual rows, \texttt{keyvaltable} also supports named row styles.

\texttt{style} = \langle \text{list of style names} \rangle \quad \text{initially: (empty)}

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

\texttt{kvtNewRowStyle\{}\langle \text{name} \rangle\}\{\langle \text{row-options} \rangle\}

This macro declares a new row style with the given \langle \text{name} \rangle and defines it to be equivalent to using the given \langle \text{row-options} \rangle. The \langle \text{name} \rangle must not already be defined.

\texttt{kvtRenewRowStyle\{}\langle \text{name} \rangle\}\{\langle \text{row-options} \rangle\}

This macro re-defines an existing row style \langle \text{name} \rangle with new \langle \text{row-options} \rangle.

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

\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}

\begin{tabular}{lll}
amount & ingredient & step \\
150g & ice cream & put into bowl \\
50g & cherries & heat up and add to bowl \\
\hline
& & serve with a smile
\end{tabular}

The \langle \text{row-options} \rangle in \texttt{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 \texttt{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}

\begin{tabular}{lll}
amount & ingredient & step \\
150g & ice cream & put into bowl \\
50g & cherries & heat up and add to bowl \\
\hline
& & serve with a smile
\end{tabular}
5.4 Cell Appearance

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

\begin{Verbatim}
\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}
\end{Verbatim}

<table>
<thead>
<tr>
<th>service</th>
<th>url</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTAN</td>
<td>ctan.org/pkg/keyvaltable</td>
</tr>
<tr>
<td>github</td>
<td>github.com/Ri-Ga/keyvaltable</td>
</tr>
<tr>
<td>Google Play</td>
<td>none</td>
</tr>
</tbody>
</table>

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}
```

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.\(^3\) 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.

\(^3\)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.
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 

\[\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}\n
The syntax for a \texttt{value} of the \texttt{headers} key in the \texttt{layout} parameter is as follows:

- \texttt{value} is a list, separated by "\"", where each element in the list specifies the columns of a single header \texttt{row}.
- Each \texttt{row}, in turn, is also a list. The elements of this list are separated by ";" (as in the columns specification of \verb|\NewKeyValTable|) and each element specifies a header \texttt{cell}.
- Each \texttt{cell} is of the form

\[
\text{\texttt{col}_1}+...+\text{\texttt{col}_n}: \text{\texttt{property}}=\text{\texttt{value}}, \text{\texttt{property}}=\text{\texttt{value}}, ...
\]

where each \texttt{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 \texttt{cell}.

The \texttt{property} pairs configure properties of the header cell. Supported \texttt{property} keys are the following.

\begin{itemize}
  \item \texttt{align} = \text{\texttt{alignment-letter}}, \text{\texttt{empty}}
  \item \texttt{initially: c}
\end{itemize}
This property specifies the alignment of content in the header cell. The \textit{value} can be set to any column alignment understood by the underlying table environment used (see Section 6.4). This particularly includes \texttt{l}, \texttt{c}, \texttt{r}, and \texttt{p}, as well as \texttt{X} for some of the table environments. The initial value can be modified with \texttt{\kvtSet{HeadCell/align=...}}.

\texttt{head = \langle text \rangle} \quad \textit{initially: \langle colspec \rangle}

This property specifies the content of the header cell. The initial value for this property is the column specification, i.e., \texttt{“\langle col\rangle+...+\langle col\rangle”}.

### 6.2 Column Spanning

The \texttt{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.

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

As the example shows, column groups are defined through the \texttt{colgroups} key of the second optional argument of \texttt{\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.

\begin{itemize}
  \item \texttt{span = \langle plus-separated columns \rangle}
  \begin{itemize}
    \item 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.
  \end{itemize}
  \item \texttt{align = \langle alignment-letter \rangle, (empty)} \quad \textit{initially: c}
  \item \texttt{format = \langle single argument macro \rangle} \quad \textit{initially: \kvtStrutted}
  \begin{itemize}
    \item 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”.
  \end{itemize}
\end{itemize}

Initial values for all the align and format options can be set with \texttt{\kvtSet}, via the \texttt{ColGroup/align} and, respectively \texttt{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.

\begin{KeyValTable}{MultiCol}
\Row{col1=1, col2=m\multicolumn{1}{r}{2}, col3=3}
\Row{col1=1, col2=m\multicolumn{2}{c}{2+3}}
\Row{col1=m\multicolumn{2}{c}{1+2}, col3=3}
\Row{col1=m\multicolumn{3}{c}{1+2+3}}
\end{KeyValTable}

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 \text{col2} is the rightmost column and therefore cannot span two columns) and also in the third row (because \text{col1} spans two columns but the second, \text{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 the backends 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}
\caption{Cherries++}
\label{Cherries}
\end{KeyValTable}
\caption{Cherries++}
\end{table}

Table \ref{Cherries} shows the recipe.

The second way to add captions is through the caption option of keyvaltable tables. This option is available for the “multipage” shape and, respectively, the table backends longtable, xltabular, and longtabu (see Section 6.4).

\begin{table}
\begin{键值表}[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}
\caption{Cherries++}
\label{Cherries}
\end{键值表}
\caption{Cherries++}
\end{table}

Table 23: Cherries++

Table 23 shows the recipe.

caption = \text{text} \hspace{1cm} \text{initially: \langle none \rangle}

caption/lot = \text{text} \hspace{1cm} \text{initially: \langle none \rangle}

caption/alt = \text{text} \hspace{1cm} \text{initially: \langle none \rangle}

label = \text{name} \hspace{1cm} \text{initially: \langle none \rangle}

These options set the caption and, respectively, label of a table. Through the option caption/lot, the caption to be put into the list of tables can be specified; if omitted, caption is used. Through the option caption/alt, the alternative
caption to be displayed on those pages of multipage tables where the main caption
is not shown; if omitted, no caption is displayed on these pages.

The position of the caption is determined by the following option.

captionpos = t, b

Initially: b

This option specifies the position of table captions. Value “t” specifies that captions
are at the top of (above) their tables; value “b” specifies that captions are at the
bottom of (below) their tables. Moreover, in case of “t” the main caption is on top
of the first page of a table while in case of “b” the main caption is at the bottom
of the last page of a table.

The following example shows the options in action.

\begin{KeyValTable}[captionpos=t, 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.

<table>
<thead>
<tr>
<th>amount</th>
<th>ingredient</th>
<th>step</th>
</tr>
</thead>
<tbody>
<tr>
<td>150g</td>
<td>ice cream</td>
<td>put into bowl</td>
</tr>
<tr>
<td>50g</td>
<td>cherries</td>
<td>heat up and add to bowl</td>
</tr>
</tbody>
</table>

Table 1 shows the recipe.

6.4 Alternative Table Environments

The keyvaltable package internally uses traditional table environments, such as
tabular, for typesetting the actual tables. Through the shape and backend
properties of a table or table type, the table environment used by for the table or,
respectively, table type can be changed. Table 2 on page 21 compares the possible
shapes/environments with regards to

- whether they support tables that span multiple pages,
- whether they support caption and label options,
- whether they support X-type (variable-width) columns,
- whether their width can be specified (through the width option), and
- whether they support a vertical or horizontal alignment of the table to be
  specified.

Finally, the table also lists the names of the packages that provide the respective
environments. The packages for the shapes onepage and multipage are loaded
automatically. All other packages must be loaded via \texttt{\usepackage} when the
respective shape or backend shall be used.

Examples can be found in Figure 2 on the next page.
Figure 2: Examples for the backend option
Table 2: Comparison of table shapes and backends

<table>
<thead>
<tr>
<th>shape</th>
<th>backend</th>
<th>multipage</th>
<th>caption</th>
<th>columns</th>
<th>width</th>
<th>align</th>
<th>packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>onepage</td>
<td>tabular/tabularx</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>v</td>
<td>tabularx</td>
</tr>
<tr>
<td>multipage</td>
<td>longtable/xltabular</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>h</td>
<td>longtable, xltabular</td>
</tr>
</tbody>
</table>

with package option compat=1.0:

<table>
<thead>
<tr>
<th>shape</th>
<th>backend</th>
<th>multipage</th>
<th>caption</th>
<th>columns</th>
<th>width</th>
<th>align</th>
<th>packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>onepage</td>
<td>tabu</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>v</td>
<td>tabu</td>
</tr>
<tr>
<td>multipage</td>
<td>longtabu</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>h</td>
<td>tabu, longtable</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>backend</th>
<th>multipage</th>
<th>caption</th>
<th>columns</th>
<th>width</th>
<th>align</th>
<th>packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabular</td>
<td></td>
<td></td>
<td></td>
<td>v</td>
<td></td>
<td></td>
</tr>
<tr>
<td>tabularx</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>v</td>
<td></td>
<td>tabularx</td>
</tr>
<tr>
<td>longtable</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>h</td>
<td></td>
<td>longtable</td>
</tr>
<tr>
<td>xltabular</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>h</td>
<td>xltabular</td>
</tr>
<tr>
<td>tabu</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>v</td>
<td></td>
<td>tabu</td>
</tr>
<tr>
<td>longtabu</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>h</td>
<td>tabu, longtable</td>
</tr>
</tbody>
</table>
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.

```latex
\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 line 1, label=two}
\end{KeyValTable}
```

<table>
<thead>
<tr>
<th>Line</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>First row</td>
</tr>
<tr>
<td>2</td>
<td>After line 1</td>
</tr>
</tbody>
</table>

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 achieved by example. The following CSV file serves as the data file in the examples. We use the same Recipe table type as previously.

```
id,amount,ingredient,step
snowman,3,balls of snow,staple all 3 balls
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
```

<table>
<thead>
<tr>
<th>id, amount, ingredient, step</th>
</tr>
</thead>
<tbody>
<tr>
<td>id, amount, ingredient, step</td>
</tr>
</tbody>
</table>

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\*.

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

<table>
<thead>
<tr>
<th>amount</th>
<th>ingredient</th>
<th>step</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>balls of snow</td>
<td>staple all 3 balls</td>
</tr>
<tr>
<td>1</td>
<td>carrot</td>
<td>stick into top ball</td>
</tr>
<tr>
<td>2</td>
<td>coffee beans</td>
<td>put diagonally above carrot</td>
</tr>
</tbody>
</table>
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`

\begin{verbatim}
expandonce = true, false
expand = true, false
\end{verbatim}

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).

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

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.

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

---

Two 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.

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

---

\begin{verbatim}
\usepackage{xint}
\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}
\end{verbatim}

---

\begin{verbatim}
\usepackage{xint}
\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}
\end{verbatim}

---

\begin{verbatim}
\usepackage{xint}
\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}
\end{verbatim}

---

\begin{verbatim}
\usepackage{xint}
\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}
\end{verbatim}

---

\begin{verbatim}
\usepackage{xint}
\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}
\end{verbatim}

---

\begin{verbatim}
\usepackage{xint}
\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}
\end{verbatim}

---

\begin{verbatim}
\usepackage{xint}
\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}
\end{verbatim}

---

\begin{verbatim}
\usepackage{xint}
\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}
\end{verbatim}

---

Two 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.

---

Two 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.

---

Two 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.

---

Two 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.

---

Two 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.

---

Two 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.

---

Two 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.

---

Two 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.

---

Two 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.

---

Two 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.

---

Two 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.

---

Two 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.
7.4 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{l}
\NewKeyValTable{Header}{
    first: head=\thead{short};
    second: head=\thead{two lines};
}\begin{KeyValTable}{Header}
\Row{first=just a, second=test}
\end{KeyValTable}
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 10.6 Row Numbering and Labeling
26  41
10.2 Auxiliary Code 10.7 Key-Value Table Content
26  41
10.3 Setting Options 10.8 Collecting Key-Value Tables
28  32
10.4 Declaring Key-Value Tables 10.9 Package Options
32  41
10.5 Custom Layout Parameters 10.10 Compatibility
35  60
10.6 Row Numbering and Labeling
41
10.7 Key-Value Table Content
41
10.8 Collecting Key-Value Tables
56
10.9 Package Options
60
10.10 Compatibility
60

10.1 Package Dependencies

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

\begin{verbatim}
\RequirePackage{etoolbox}
\RequirePackage{xkeyval}
\RequirePackage{trimspaces}
\end{verbatim}

We use \texttt{booktabs} for nice horizontal lines and \texttt{xcolor} for row coloring.

\begin{verbatim}
\PassOptionsToPackage{table}{xcolor}
\RequirePackage{xcolor}
\RequirePackage{booktabs}
\end{verbatim}

10.2 Auxiliary Code

10.2.1 List Parsing

The \texttt{\textbackslash kvt@DeclareTrimListParser\{\langle\texttt{command}\rangle\}\{\langle\texttt{separator}\rangle\}} macro is equivalent to \texttt{etoolbox}'s \texttt{\textbackslash \texttt{DeclareListParser}}, except that the \langle\texttt{command}\rangle is defined such that it will remove trailing spaces from list elements before passing the list elements to the processing macro (i.e., to \texttt{\textbackslash do} or the user-provided macro). Note: With \texttt{\textbackslash \texttt{DeclareListParser}}, \langle\texttt{command}\rangle is defined to only remove leading spaces but not trailing ones. This implementation relies on the internals of \texttt{etoolbox} and works with v2.4 of the package, at least.

\begin{verbatim}
\newcommand\kvt@DeclareTrimListParser{%\ifstar{\kvt@DeclareTrimListParser@i{*}}{\kvt@DeclareTrimListParser@i{}}\DeclareListParser#1{#2}{#3}\expandafter\patchcmd\csname etb@lst@\expandafter\@gobble\string#2\endcsname{\etb@listitem}{\kvt@etb@listitem}{}\kvt@warn{Failed to patch a command defined by the etoolbox package, possibly because etoolbox internals have changed. You might encounter superfluous spaces.}}}
\end{verbatim}

26
The cascade of \expandafter below ensures that first the trimming macro is expanded and afterwards the outer \unexpanded of the timing macro's expansion is expanded, which by definition of the “noexp” trimming macro fully expands the macro's logic. The auxiliary macro below is only for switching the two arguments such that the expansion control can be applied to the second argument.

\newcommand\kvt@etb@listitem[2]{%\expandafter\expandafter\expandafter\kvt@etb@listitem@i\expandafter\expandafter\expandafter{\trim@post@space@noexp{#2}}{#1}}
\newcommand\kvt@etb@listitem@i[2]{\etb@listitem{#2}{#1}}
\kvt@dossvlist The \kvt@dossvlist\{\textsc{list}\} macro parses a semicolon-separated list and runs \texttt{\textbackslash do\{item\}} for every element of the list.
\newcommand\kvt@forpsvlist\kvt@forpsvlist\{\textsc{handler}\}\{\textsc{list}\} parses a '+'-separated list.
\newcommand\kvt@forpsvlist\kvt@forpsvlist\{\textsc{handler}\}\{\textsc{list}\}{+}
\newcommand\kvt@dobrklist The \kvt@dobrklist\{\textsc{list}\} parses a '\-'-separated list.
\kvt@DeclareTrimListParser\kvt@forpsvlist\{+\}
\kvt@DeclareTrimListParser\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@Colsetkeys The \kvt@setkeys\{\textsc{keys}\}\{\textsc{family}\} macro abbreviates \setkeys[kvt]{\textsc{family}}\{\textsc{keys}\} (note the reverse order of arguments). The \kvt@setcmdkeys\{\textsc{keycmd}\}\{\textsc{family}\} and \kvt@setcskeys\{\textsc{keycs}\}\{\textsc{family}\} abbreviate the cases where \textsc{keys} are stored in macro \textsc{keycmd} or, respectively, stored in a macro with name \textsc{keycs}.
\newcommand\kvt@setkeys[2]{\setkeys[kvt]{#2}{#1}}
\newcommand\kvt@setcmdkeys[2]{\expandafter\kvt@setkeys\expandafter{#1}{#2}}
\newcommand\kvt@setcskeys[2]{\expandafter\kvt@setcmdkeys\expandafter{\csname #1\endcsname}{#2}}
The \kvt@setkeys@nopresets\{\textsc{keys}\}\{\textsc{family}\} macro expands to a \kvt@setkeys in which no presets are active.
\newcommand\kvt@setkeys@nopresets[2]{%\expandafter\kvt@xkv@disablepreset[kvt]{#2}{\kvt@setkeys{#1}{#2}}}
\kvt@Colsetkeys The \kvt@colsetkeys\{\textsc{family}\}\{\textsc{keys}\} macro abbreviates \setkeys[KeyValTable]{\textsc{family}}\{\textsc{keys}\} with the same arguments. The \kvt@colsetcmdkeys\{\textsc{famcmd}\}\{\textsc{keys}\} and \kvt@colsetcskeys\{\textsc{famcs}\}\{\textsc{keys}\} abbreviate the cases where \textsc{family} is stored in macro \textsc{famcmd} or, respectively, stored in a macro with name \textsc{famcs}.
\newcommand\kvt@colsetkeys[2]{\setkeys[KeyValTable]{\textsc{family}}\{#1\}\{#2\}}
\newcommand\kvt@colsetcmdkeys[2]{%
The \kvtStrutted\{(inner)\}\{(arg)\} macro prefixes and suffixes the argument \langle arg\rangle with a \textbackslash 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 \langle arg\rangle, \langle inner\rangle\{(arg)\} is prefixed and sufficed with \textbackslash strut.

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

10.3 Setting Options

The \kvtSet\{(options)\} set the default options, which apply to all tables typeset with the package.

\newcommand\kvtSet[1]{\kvt@setkeys{#1}{global,Table,Column}\ifdefvoid\kvt@@presetqueue{}{\kvt@@presetqueue\undef\kvt@@presetqueue}}

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

\newcommand\kvt@@lazypreset[2]{\appto\kvt@@presetqueue{\presetkeys[kvt]{#1}{#2}{}}}
The following options only abbreviate options defined above.

\define@boolkey[kvt]{Table}{norowbg}[true]{%}
\define@boolkey[kvt]{Table}{nobg}[true]{%}
\define@boolkey[kvt]{Table}{norules}[true]{%}
\define@key[kvt]{Table}{backend}{
  \ifinlist{#1}{\kvt@@tablebackends}
  \csdef{cmdkvt@Table@shape}{#1}
  \kvt@error{Table backend '#1' not known}
  \kvt@error{Check for misspellings in '#1'}}}
\define@key[kvt]{Table}{shape}{
  \ifinlist{#1}{\kvt@@tableshapes}
  \csdef{cmdkvt@Table@shape}{#1}
  \ifinlist{#1}{\kvt@@tablebackends}
  \kvt@warn{Using a backend ('#1') as shape is deprecated.}
  \Use the 'backend' option instead.\%
  \csdef{cmdkvt@Table@shape}{#1}
  \kvt@error{Table shape '#1' not known}
  \kvt@error{Check for misspellings in ' #1'})}}

The following table options only apply to individual KeyValTable environments and cannot be set with \NewKeyValTable or \kvtSet.

\define@cmdkey[kvt]{TableEnv}{caption}{}
\define@cmdkey[kvt]{TableEnv}{caption/lot}{}
\define@cmdkey[kvt]{TableEnv}{caption/alt}{}
\define@cmdkey[kvt]{TableEnv}{label}{}
\define@boolkey[kvt]{TableEnv}{resume}[true]{%}
\define@boolkey[kvt]{TableEnv}{resume*}[true]{%}
\define@boolkey[kvt]{TableEnv}{rowcountlast}{%}
\ifbool{#1}{%}
  \kvt@error{No previous table whose counter could be resumed.}
  \Check whether the "resume" is intentional and whether a
  previously existing predecessor table has disappeared.\{}){}){}
\else{}%}
  \ifundef{\kvt@@rowcountlast}
    \kvt@error{No previous table whose counter could be resumed.}
    \Check whether the "resume" is intentional and whether a
    previously existing predecessor table has disappeared.\}()}\{}){}
\fi
\define@boolkey[kvt]{TableEnv}{resume*}[true]{%}
\ifbool{#1}{%}
  \kvt@error{No previous table whose counter could be resumed.}
  \Check whether the "resume*" is intentional and whether a
  previously existing predecessor table has disappeared.\})}{}
\else{}%}
The \kvt@@lastenvopt macro holds the previous KeyValTable’s options. Beyond these options, resume* automatically also sets resume.
\ifundef{\kvt@@lastenvopt}{%}
\texttt{\kvt@error{No previous table whose options could be resume*’d.}}
\texttt{Check whether the “resume*” is intentional and whether a previously existing predecessor table has disappeared.}}\%\kvt@setcmdkeys\setkeys{resume}{TableEnv}\%

\subsection*{10.3.2 Column Options}
The following code defines the possible column options.
\begin{verbatim}
\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}}
\end{verbatim}

\texttt{The \kvt@colkeysetter{⟨key⟩}{{⟨value⟩}} specializes \kvt@keysetter for column options.}
\begin{verbatim}
\newcommand\kvt@colkeysetter[2]{\kvt@keysetter{\kvt@@column}{Column}{#1}{#2}{\csdef{kvt@col@#1@\kvt@@column}{#2}}}
\end{verbatim}

\texttt{When used in \kvtSet, this key sets the preset value for the ⟨key⟩ in ⟨family⟩.}
\begin{verbatim}
\define@cmdkey[kvt]{ColGroup}{span}{\csgdef{kvt@colgrp@span@\kvt@@tname @\kvt@@colgrp}{#1}}
\define@cmdkey[kvt]{ColGroup}{align}{\csgdef{kvt@colgrp@align@\kvt@@tname @\kvt@@colgrp}{#1}}
\define@cmdkey[kvt]{ColGroup}{format}{\csgdef{kvt@colgrp@format@\kvt@@tname @\kvt@@colgrp}{#1}}
\kvt@def@globalopts{ColGroup}{align, format}
\end{verbatim}

\subsection*{10.3.3 Layout Customization Options}
The following defines the option keys for the second optional argument to \texttt{\NewKeyValTable}. These options intentionally do not support setting global defaults via \texttt{\kvtSet}.
\begin{verbatim}
\define@cmdkey[kvt]{Layout}{headers}{\expandafter\kvt@parseheadrows\expandafter{\kvt@@tname}{#1}}
\define@cmdkey[kvt]{Layout}{colgroups}{\expandafter\kvt@parsecolgroups\expandafter{\kvt@@tname}{#1}}
\end{verbatim}

\begin{verbatim}
\define@cmdkey[kvt]{ColGroup}{span}{\csgdef{kvt@colgrp@span@\kvt@@tname @\kvt@@colgrp}{#1}}
\define@cmdkey[kvt]{ColGroup}{align}{\csgdef{kvt@colgrp@align@\kvt@@tname @\kvt@@colgrp}{#1}}
\define@cmdkey[kvt]{ColGroup}{format}{\csgdef{kvt@colgrp@format@\kvt@@tname @\kvt@@colgrp}{#1}}
\kvt@def@globalopts{ColGroup}{align, format}
\end{verbatim}

\begin{verbatim}
\define@cmdkey[kvt]{Layout}{headers}{\expandafter\kvt@parseheadrows\expandafter{\kvt@@tname}{#1}}
\define@cmdkey[kvt]{Layout}{colgroups}{\expandafter\kvt@parsecolgroups\expandafter{\kvt@@tname}{#1}}
\end{verbatim}
The following defines the options for header cells.

10.3.4 Row Options

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

10.3.5 Option Defaults

The following sets the default values for the options. This is done only after the package is otherwise completely processed, to ensure that all features are already defined/registered at that point.
10.4 Declaring Key-Value Tables

\NewKeyValTable \[ \langle \text{options} \rangle \] \{ \langle \text{tname} \rangle \} \{ \langle \text{colspecs} \rangle \} \[ \langle \text{layout} \rangle \] declares a new key-value table type, identified by the given \langle \text{tname} \rangle. The columns of the table type are specified by \langle \text{colspecs} \rangle. The optional \langle \text{options} \rangle, if given, override the default table options for tables of type \langle \text{tname} \rangle.

Before doing anything, check whether \langle \text{tname} \rangle has already been defined.

First initialize the “variables”.

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 column of tables is that this code would prevent \multicolumn to be used in the first column.

\csdef{kvt@alignments@#2}{\%}
\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{%\%}

Catch syntax errors first.

\kvt@checkcolspecempty{#4}{column}{#2}%
\kvt@checkcolumnequal{#2}%
\trim@spaces@in\kvt@@column
\expandafter\kvt@parsecolspec@i\expandafter{\kvt@@column}{#1}{#3}%

Check and record the column name first.

\ifinlists{#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.

\ifs@string{kvt@col@hidden@#1@#2}{true}{\%}
\csappto{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.

```
\ndefault\kvt@column{\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@defaulthead` yields “⟨rowcolor⟩⟨fmthead1⟩ & … & ⟨fmtheadn⟩\tabularnewline”. In the above, ⟨rowcolor⟩=⟨rowcolor⟩⟨⟨headbg⟩⟩.

```
\newcommand\kvt@defaulthead{\noexpand\kvt@rowcolorornot{\cmdkvt@Table@headbg}\kvt@defaulthead@i{}\
\newcommand\kvt@defaulthead@i[2]{\kvt@ifnil{#2}{\noexpand\tabularnewline}{\unexpanded{#1}\
\ifdefvoid\cmdkvt@Table@headalign{\expandonce\cmdkvt@Table@headformat{\unexpanded{#2}}}\
\noexpand\multicolumn{1}{\expandonce\cmdkvt@Table@headalign}{\expandonce\cmdkvt@Table@headformat{\unexpanded{#2}}}\
\kvt@defaulthead@i{&}}}\
```

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]{}\
\newcommand\kvt@ifnil[1]{\ifx⟨val⟩@nil\relax\expandafter\@firstoftwo\else\expandafter\@secondoftwo\fi}\
```
The \kvt@alltables is an etoolbox list containing the names of all tables declared by \NewKeyValTable.

\begin{verbatim}
\newcommand\kvt@alltables{}
\end{verbatim}

\section{Custom Layout Parameters}

The \kvt@parselayout{\langle layout-opts \rangle}{\langle tname \rangle} macro parses the layout options, \langle layout-opts \rangle, for table type \langle tname \rangle.

\newcommand\kvt@parselayout[2]{\%
\def\kvt@@tname{#2}\%
Now parse the \langle layout-opts \rangle. The keys are defined such that their handlers already do the parsing.
\kvt@setkeys{#1}{Layout}\%
\undef\kvt@@tname}

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

\newcommand\kvt@parsecolgroups[2]{\%
\begingroup\kvt@@result collects the parsing outcome code that shall escape the group started above.
\kvt@setkeys{#1}{ColGroup}\%
\kvt@checkcolgroupcs{kvt@colgrp@span@#1@#2}{#1}{#2}\%
The following defines the \Row key for \langle cgname \rangle, as an abbreviation for setting the value of the first displayed column of \langle cgname \rangle (\kvt@colgrp@first to a \multicolumn that spans the “right” number of columns).
\eappto\kvt@@result{\%
\noexpand\define@cmdkey[KeyValTable]{#1}{#2}{\%
\end{verbatim}
The following `\ifdefvoid` check ensures that if \( \langle cgname \rangle \) is a hidden column group (i.e., a column group of which all spanned columns are hidden), then setting \( \langle cgname \rangle \) to a value has no effect.

The “abbreviation” is implemented via `\setkeys`. The letter normally employs the defined `\presetkeys`, but we disable this through `\kvt@xkv@disablepreset` to avoid that column keys that are set before a colgroup key are overwritten by their preset values.

Notice the “*” after `\kvt@colgrp@first`, which disables the first column’s default formatting to replace it by the formatting of \( \langle cgname \rangle \).

The `\kvt@checkcolspecempty` macro checks the \( \langle empty \rangle \) parameter of a parsing macro for a colon-separated key-value pair. If \( \langle empty \rangle \) is empty, this corresponds to the valid case that only the name of a column group was provided and no properties. If \( \langle empty \rangle \) equals “:”, then a name and properties were provided. In all other cases, superfluous colons were found.

The `\kvt@checkcolgroup` macro performs some checks on \( \langle span-psv \rangle \) as a specification of which columns shall be spanned by a group column of name \( \langle cgname \rangle \). The checks are

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

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`.
First, check individual columns in \texttt{⟨span-psv⟩} and transfer them into a “map”, \texttt{kvt@incolgrp} that simply records which column names occur in \texttt{⟨span-psv⟩}.

```latex
\def\kvt@@psvdo##1{% 
  \ifinlistcs{##1}{kvt@allcolumns@#2}{}\kvt@error
  \{Column `##1' referenced in column group `#3' not known \}
  \{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”.

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

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

```latex
\def\kvt@@status{0}%
\kvt@@coldo{⟨column⟩} is applied to each displayed column, in order.
```

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

```latex
\ifcsvoid{kvt@@incolgrp@##1}{}
\ifcase\kvt@@status \or
  \def\kvt@@status{2}\fi%
\edef\kvt@@colgrp@n{\the\numexpr\kvt@@colgrp@n+1\relax}%
```

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

```latex
\ifexpandafter\ifcase\kvt@@status
  \def\kvt@@status{i}\def\kvt@@colgrp@first{#1}\or
  \kvt@error\{Column group `\kvt@@colgrp@first' must consist of only \texttt{⟨column⟩} as \texttt{kvt@@colgrp@first}; if \texttt{kvt@@status} is previously 2, then the columns in \texttt{⟨span-psv⟩} would not be consecutively displayed and, hence, an error is raised.\}%
  \{Compare the \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}%
```

This macro is not encapsulated in a group (in order to return \texttt{kvt@@colgrp@n} and \texttt{kvt@@colgrp@first}, we finally prevent the local \texttt{kvt@incolgrp@⟨column⟩} from leaking outside this macro.

```latex
\csundef{kvt@@incolgrp@##1}%
\forlistcsloop{\kvt@@coldo}{kvt@displaycols@#2}%
```
The `\kvt@checkcolgroupcs{(span-psw-cs)}{(tname)}{(cname)}` 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.

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

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`.

\begin{verbatim}
\newcommand\kvt@parseheadrows[2]{%
  \iftstrempy{#2}{}{\kvt@parseheadrows@i{#2}{#1}}%
\newcommand\kvt@parseheadrows@i[2]{%
  \csdef{kvt@@custheadrows@#2}{}%
  \csdef{kvt@headrowcount@#2}{0}%
  \begingroup
    \def\kvt@@parseheadrows{}%
    \do\kvt@parseheadrow{#1}{#2}%
    \csletcs{kvt@headings@#2}{kvt@@custheadrows@#2}}%
\end{verbatim}

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.

\begin{verbatim}
\def\do##1{%\ifstrequal{##1}{::}{\appto\kvt@@parseheadrows{\cseappto{kvt@@custheadrows@#2}{\csexpandonce{kvt@headings@#2}}}}{%\appto\kvt@@parseheadrows{\kvt@parseheadrow{#2}{##1}}}%\appto\kvt@@parseheadrows{\csedef{kvt@headrowcount@#2}{\the\numexpr\csuse{kvt@headrowcount@#2}+1\relax}}}%\kvt@dobrklist{#1}%
\end{verbatim}

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

\begin{verbatim}
\expandafter\endgroup\kvt@@parseheadrows\csletcs{kvt@headings@#2}{kvt@@custheadrows@#2}}%
\end{verbatim}

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

\begin{verbatim}
\newcommand\kvt@parseheadrow[2]{%
  \begingroup
    \kvt@parsehdcolspec{#1}{#2}%
  \endgroup
\end{verbatim}

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).

\begin{verbatim}
\def\do##1{\kvt@parsedhcolspec{#1}{#2}{#1}{\undefined}}%
\end{verbatim}
Initialize variables for the subsequent loop. The \texttt{\kvt@@tmpgrphd} macro collects the code for the cells of the current header row. The \texttt{\kvt@@span} counter specifies how many columns the current cell shall span. Finally, \texttt{\kvt@@curhd} and \texttt{\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.

\begin{verbatim}
350 \let\kvt@@tmpgrphd\@empty
351 \kvt@@span\z@
352 \undef\kvt@@curhd \undef\kvt@@lasthd
353 \kvt@def@atseconduse\kvt@@switchcol{% 
354 \do\langle\colname\rangle 
355 \ifdefequal\kvt@@curhd\kvt@@lasthd 
356 \else 
357 \expandafter\kvt@concludehdcolumn
358 \edef\do{\noexpand\csappto{kvt@@custheadrows@\tname}{% 
359 \unexpanded{\noexpand\kvt@rowcolorornot{\cmdkvt@Table@headbg}}% 
360 \noexpand\unexpanded{\expandonce{\kvt@@tmpgrphd}}}}%
361 \expandafter\endgroup\do}
\end{verbatim}

Next, loop over all displayed columns, stored in \texttt{\kvt@displaycols@\tname}. The following \texttt{\do\langle\colname\rangle} macro collects (spanned) columns as specified in \texttt{\colspec}, in the ordering in which the table’s columns are displayed. The spanned columns are stored in \texttt{\kvt@@tmpgrphd}.

\begin{verbatim}
354 \def\do##1{\letcs\kvt@@curhd{kvt@@hdcellof@##1}% 
355 \ifdefequal\kvt@@curhd\kvt@@lasthd 
356 \else 
357 \expandafter\kvt@concludehdcolumn
358 \ifdefvoid\kvt@@curhd{}{\ifcsdef{kvt@@hdcelldone@\kvt@@curhd}{}{}}%
359 \kvt@@span\@ne \let\kvt@@lasthd\kvt@@curhd}%
360 \dolistcsloop{kvt@displaycols@\tname}%
\end{verbatim}

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

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 \texttt{\kvt@@curhd}) and set \texttt{\kvt@@lasthd} to the current one.

\begin{verbatim}
357 {\ifnum\kvt@@span\z@ \expandafter\kvt@concludehdcolumn\fi 
358 \ifdefvoid\kvt@@curhd{}{\ifcsdef{kvt@@hdcelldone@\kvt@@curhd}{}{}}%
359 \kvt@error{Header cell `\kvt@@curhd' must consist of only 
360 consecutive columns, but it is not}%
361 {Compare `\string\kvt@@curhd' to the column ordering as 
362 specified in `\string\NewKeyValTable{#1}'}}{}
363 \kvt@@span\@ne \let\kvt@@lasthd\kvt@@curhd%
364 }\dolistcsloop{kvt@displaycols@\tname}%
365 \kvt@concludehdcolumn
\end{verbatim}

Finally, conclude the whole header row and append the row to the overall list of rows, stored in \texttt{\kvt@@custheadrows@\tname}, while ending the current \LaTeX group.

\begin{verbatim}
366 \expandafter\endgroup\do}
367 \kvt@rowcolorornot The \texttt{\kvt@rowcolorornot\langle\color\rangle} expands to \texttt{\rowcolor\langle\color\rangle} if \langle\color\rangle is nonempty and does have no effect if \langle\color\rangle is empty.

\texttt{\kvt@rowcolorcmdornot} The \texttt{\kvt@rowcolorcmdornot\langle\cmd\rangle} expands to \texttt{\rowcolor\langle\cmd\rangle}, where \langle\cmd\rangle is the (one-time) expansion of \langle\cmd\rangle, if \langle\cmd\rangle is a defined macro whose expansion is nonempty; its expansion is empty otherwise.

\begin{verbatim}
371 \newcommand{\kvt@rowcolorornot}{\ifstreempty{\cmd}{\rowcolor{\color}{\cmd}}}%
372 \newcommand{\kvt@rowcolorcmdornot}{\ifdefempty{\cmd}{\rowcolor{\color}{\cmd}}}%
373 \expandafter\rowcolor\expandafter[\cmd]
\end{verbatim}

39
The counter \texttt{\kvt@@bodyrow} is used internally in KeyValTable environments for keeping track of rows for the background-coloring. The difference between this counter and \texttt{\kvtRow} is that the former also counts uncounted rows and is unaffected by the \texttt{resume} option. The counter only counts rows produced by \texttt{\Row} and its corresponding collecting counterparts. Header rows as well as manually inserted rows, including those produced by macros like \texttt{\midrule} in a \texttt{longtable} environment, are not counted (as opposed by the internal counter of \texttt{\rowcolors}.

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

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

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.

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

Next, 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.
10.6 Row Numbering and Labeling

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

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

\begin{verbatim}
\newcounter{kvtRow}
\end{verbatim}

\text{kvtTypeRow} The \text{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 \text{\kvt@rowcount@⟨tname⟩} macro, which keeps track of the individual row counts of the \textlangle tname\rangle type.

\begin{verbatim}
\newcounter{kvtTypeRow}
\end{verbatim}

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

\begin{verbatim}
\newcounter{kvtTotalRow}
\setcounter{kvtTotalRow}{0}
\end{verbatim}

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

\begin{verbatim}
\newcommand\kvtLabel[3][{}]{%\%}
\end{verbatim}

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

\begin{verbatim}
\setcounter{kvt@LabelCtr}{\value{#2}}\%
\addtocounter{kvt@LabelCtr}{-1}\%
\refstepcounter{kvt@LabelCtr}\%
\end{verbatim}

Next, define the \text{⟨label⟩} (if provided) and show the value of \text{⟨counter⟩}.

\begin{verbatim}
\ifstrempty{#3}{}{\%}
\ifstrempty{#1}{}{\%}
\end{verbatim}

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

\begin{verbatim}
\newcounter{kvt@LabelCtr}
\end{verbatim}

10.7 Key-Value Table Content

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

\begin{verbatim}
\newenvironment{KeyValTable}[2][]{%
\end{verbatim}
The \Row\{\langle options\rangle\}\{\langle content\rangle\} macro is made available locally in the KeyValTable environment.

413 \def\Row\{\kvt@AddKeyValRow
414 \langle\noalign{\bgroup}\langle\\expandafter\egroup\kvt@@row\rangle\#2\rangle\%\kvt@SetOptions{\#2}{\#1}\%

Save \langle options\rangle globally for a potential “resume*” option in the immediately following KeyValTable environment. If resume* is specified for the current environment, then the previous options are not replaced. That is, resume* resumes the options from the previous non-resuming environment.

416 \ifbool{\kvt@TableEnv@resume*}{\}
417 \{\gdef{\kvt@@lastenvopt}{\#1}\%
418 \csuse{\kvt@StartTable@\cmdkvt@Table@shape}{\#2}\%
419 \}\%
420 \csuse{\kvt@EndTable@\cmdkvt@Table@shape}\}

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

421 \newcommand{\kvt@SetOptions@i}[2]{\expandafter\kvt@SetOptions@i\expandafter{\csname kvt@options@#1\endcsname}{#2}}

The auxiliary macro \kvt@SetOptions@i\{\langle optcmd\rangle\}\{\langle options\rangle\} first sets the options in the expansion of \langle optcmd\rangle and then the \langle options\rangle.

423 \newcommand{\kvt@SetOptions@i}[1][2]{\expandafter\kvt@SetOptions@i\expandafter{\#1,\#2}{Table,TableEnv}}

10.7.1 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@boolkey[metatbl]{EnvProp}{canVAlign}{\%}
432 \metatbl@boolprop{canVAlign}{\#1}\%
433 \define@boolkey[metatbl]{EnvProp}{canHAlign}{\%}
434 \metatbl@boolprop{canHAlign}{\#1}\%
435 \define@cmdkey[metatbl]{EnvProp}{packages}{\\metatbl@setprop\{pkg\}{\#1}}

The atEnd property shall be set to \LaTeX 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.

436 \define@key[metatbl]{EnvProp}{atEnd}{\\metatbl@setprop\{atEnd\}{\#1}}
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.

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⟩`.

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

All macros in the following block expect three arguments, `{⟨env-name⟩}{⟨iftrue⟩}{⟨iffalse⟩}`. The macro `\metatblRegistered` expands to `⟨iftrue⟩` if `⟨env-name⟩` has been registered via `\metatblRegisterEnv` and expands to `⟨iffalse⟩` otherwise. The remaining macros expand to `⟨iftrue⟩`, if the respective property has been set to true in when `⟨env-name⟩` was registered via `\metatblRegisterEnv`, and expand to `⟨iffalse⟩` otherwise.

Macros `\metatblUsePackage{⟨env-names⟩}` and `\metatblRequire{⟨env-names⟩}` 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.
The auxiliary macro \texttt{metatbl@csnamearg\{⟨command⟩\}}\{⟨csname⟩\} passes the expansion of the macro with name \texttt{⟨csname⟩} as the first argument to \texttt{⟨command⟩}.

The following are the properties of some basic table environments.

\begin{verbatim}
\metatblRegisterEnv{tabular}{
  isLong=false, hasWidth=false, isTabu=false, hasCaption=false,
  canVAlign=true, canHAlign=false,
  packages={},
  atEnd={\preto\endtabular{#1}},
}
\metatblRegisterEnv{tabularx}{
  isLong=false, hasWidth=true, isTabu=false, hasCaption=false,
  canVAlign=true, canHAlign=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,
  canVAlign=false, canHAlign=true,
  packages={longtable},
  atEnd={\preto\endlongtable{#1}},
}
\metatblRegisterEnv{xltabular}{
  isLong=true, hasWidth=true, isTabu=false, hasCaption=true,
  canVAlign=false, canHAlign=true,
  packages=xltabular,
  atEnd={\preto\XLT@ii@TX@endtabularx{\toks@\expandafter{\the\toks@#1}}},
}
\metatblRegisterEnv{tabu}{
  isLong=false, hasWidth=true, isTabu=true, hasCaption=false,
  canVAlign=true, canHAlign=false,
  packages={tabu},
  atEnd={\preto\endtabular{#1}},
}
\metatblRegisterEnv{longtabu}{
  isLong=true, hasWidth=true, isTabu=true, hasCaption=true,
  canVAlign=false, canHAlign=true,
  packages={tabu,longtable},
}
\end{verbatim}

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

\begin{longtabu}...\end{longtabu}

### 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.

\begin{verbatim}
\newcommand\kvt@StartTabularlike[2]{
  \metatblAtEnd{#1}{\kvt@@endhook}
  \let\kvt@@endhook\@empty
  \let\kvt@@prehook\@empty
  \ifbool{kvt@Table@showrules}
    {\def\kvt@@rule##1{\csuse{##1rule}}}
    {\def\kvt@@rule##1{}}
  \appto\kvt@@prehook{\kvt@@rule{top}}
  \appto\kvt@@endhook{\kvt@@rule{bottom}}
  \csxdef{kvt@rowcount@#2}{\thekvtTypeRow}
  \csxdef{kvt@@rowcountlast}{\thekvtRow}
\end{verbatim}

The following saves the row counter value for the table type globally, such that subsequent tables of the same `⟨tname⟩` can start counting from there. Moreover, it save the local row counter for the case that the next table uses the "resume" option.

\begin{verbatim}
\appto\kvt@@endhook{\noalign{\csxdef{kvt@rowcount@#2}{\thekvtRow}}}
\end{verbatim}

4This hooking into `\@mkpream` is inspired by how `tabularx` replaces `X` columns by `p` columns as part of its measuring.
Adding caption and label.

523 \ifdefvoid\cmdkvt@TableEnv@caption
524 \let\kvt@caption@main@empty
525 \let\kvt@caption@alt@empty
526 \metatblHasCaption{#1}
527 \ifdef\kvt@TableEnv@caption\main
528 \csexpandonce{kvt@caption@\cmdkvt@Table@captionpos}\%
529 \ifdefvoid\cmdkvt@TableEnv@caption/lot}{
530 \expandonce\cmdkvt@TableEnv@caption
531 \ifdefvoid\cmdkvt@TableEnv@label{}{\label{\expandonce\cmdkvt@TableEnv@label}}
532 \noexpand\%
533 \noexpand{}% \noexpand\%
534 \ifdefvoid\cmdkvt@TableEnv@caption/alt}{
535 \ifdefstring{\cmdkvt@Table@captionpos}{t}{
536 \let\kvt@@caption@headmain\kvt@@caption@main
537 \let\kvt@@caption@footmain\@empty
538 \let\kvt@@caption@headalt\kvt@@caption@alt
539 \let\kvt@@caption@footalt\@empty
540 \noexpand\%
541 \noexpand\%}
542 \kvt@error{Caption lost, table backend '#1' does not support captions}
543 {Consider placing the KeyValTable environment inside a 'table'
544 environment and use the \string\caption space macro inside.}%%
545 \ifdefstring{\cmdkvt@Table@captionpos}{t}{
546 \let\kvt@@caption@headmain\kvt@@caption@main\let\kvt@@caption@footmain\@empty
547 \let\kvt@@caption@headalt\kvt@@caption@alt\let\kvt@@caption@footalt\@empty
548 \let\kvt@@caption@footmain\kvt@@caption@main\let\kvt@@caption@headmain\empty
549 \let\kvt@@caption@footalt\kvt@@caption@alt\let\kvt@@caption@headalt\empty
550 \noexpand\%
551 \noexpand\%}
552 \}%
553 \% The following lines perform some checks before the table environment is started.
554 \ifdefvoid\cmdkvt@Table@valign\{\metatblCanVAlign{#1}\%
555 \kvt@warn{Table environment '#1' of table '#2'
556 does not support the vertical alignment option (valign).
557 Ignoring the option}%%
558 \ifdefvoid\cmdkvt@Table@halign\{\metatblCanHAlign{#1}\%
559 \kvt@warn{Table environment '#1' of table '#2'
560 does not support the horizontal alignment option (halign).
561 Ignoring the option}%%
562 \% Initializing the row counters. The global counter kvtTotalRow needs no local initialization.
563 \global\kvt@bodyrow=\relax
564 \ifbool{\kvt@TableEnv@resume}{
565 \setcounter{kvtRow}{\csuse{kvt@@rowcountlast}}
566 \setcounter{kvtTypeRow}{\csuse{kvt@rowcount@#2}}
567 \noexpand\%
568 \noexpand\%}
569 \noexpand\%
570 \noexpand\% Initialize the background colors for the body rows.
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.

As background on the positions of the parameters below, here is the syntax for beginning the supported environments:

- \begin{tabular}{{\textalign}}{{\preamble}}
- \begin{tabularx}{{\width}}{{\textalign}}{{\preamble}}
- \begin{longtable}{{\textalign}}{{\preamble}}
- \begin{xltabular}{{\textalign}}{{\width}}{{\preamble}}
- \begin{tabu}to \width{{\textalign}}{{\preamble}}
- \begin{longtabu}to \width{{\textalign}}{{\preamble}}

The above cases are covered in the following lines.

The remainder below already starts the content of the table environment. It also sets the header and footer for multipage tables.

The \kvt@caption@b and \kvt@caption@t macros behave like \caption but add extra behavior depending on whether the caption is displayed above (\kvt@caption@t) or below (\kvt@caption@b) the table. Currently, \kvt@caption@b only fixes the spacing between the table and the caption.

\newcommand{\kvt@caption@b}{}\newcommand{\kvt@caption@t}{\caption}

\newcommand{\kvt@caption@b}{\caption}
\newcommand{\kvt@caption@t}{\caption}

Fixme: The following \baselineskip before the caption compensates that \longtable adds a \baselineskip below the caption (in its macro \LT@makecaption) but
not above. The \ltcaption package replaces the hard-coded \baselineskip by \LTcapskip but also only puts it below the caption. The code below could at least be improved to use \LTcapskip if it is available.

The \kvt@setrowcolors{\{colors\}} sets up row colors using the \rowcolors macro of xcolor. The \{colors\} parameter expects arguments of the form \langle color1\rangle..\langle color2\rangle (the syntax used for the rowbg option). The row colors then alternate between \langle color1\rangle and \langle color2\rangle, starting with \langle color1\rangle in the first row. If \{colors\} is empty, then no row colors are setup.

The \kvt@userowcolors macro expands to \rowcolor{\langle color\rangle}, where \langle color\rangle is the background color set via \kvt@setrowcolors for odd, respectively even rows, based on \kvt@@bodyrow.

The \kvt:RegisterBackend{\{env\}} macro registers the table environment \{env\} as a table backend for use by keyvaltable. The \kvt:RegisterShape{\{name\}}{\{nonX-env\}}{\{X-env\}} registers a shape with the given \{name\} and associates it with the environment \langle nonX-env\rangle when the shape is used for a table without X columns and with environment \langle X-env\rangle otherwise.
The macros \kvt@@tablebackends and \kvt@@tableshapes hold etoolbox lists of registered names of table backends and table shapes.

\newcommand\kvt@@tablebackends{}
\newcommand\kvt@@tableshapes{}

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.

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

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 \textit{X} columns and that table environment ⟨X-env⟩ is used for typesetting tables that do use \textit{X} columns.

\newcommand\kvt@DefineDualTabEnv[3]{\expandafter\expandafter\expandafter\metatbl@ifhasXcolumns\expandafter\expandafter\expandafter{\csname kvt@alignments@#1\endcsname}}

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

\newcommand\kvt@ifhasXcolumns[1]{\expandafter\expandafter\expandafter\metatbl@ifhasXcolumns\expandafter\expandafter\expandafter{\csname kvt@alignments@#1\endcsname}}

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

\kvt@RegisterBackend{tabular}
\kvt@RegisterBackend{longtable}
\kvt@RegisterBackend{tabularx}
\kvt@RegisterBackend{xtabular}
\kvt@RegisterBackend{tabu}
\kvt@RegisterBackend{longtabu}

10.7.3 Environment-Independent Parts

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 \langle pre \rangle and \langle post \rangle 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%
\Ifnextchar[\kvt@AddKeyValRow@i{#2}{#3}}
\kvt@AddKeyValRow@i{\langle post \rangle}{\langle tname \rangle}{\langle options \rangle}{\langle content \rangle} macro parses \langle options \rangle and evaluates the hidden option.

\def\kvt@AddKeyValRow@ii#1#2#3{% Initialize and first add the \noalign material to the row.
\def\kvt@AddKeyValRow@ii{\langle post \rangle}{\langle tname \rangle}{\langle content \rangle} macro mainly processes \langle content \rangle as well as \langle options \rangle that have already been parsed by \kvt@AddKeyValRow@i.

\ifdef\cmdkvt@Row@align{\let\kvt@rowmkmulticolumn\@empty}
\edef\kvt@rowmkmulticolumn{\noexpand\multicolumn{1}{\expandonce\cmdkvt@Row@align}}%
\Ifdef\cmdkvt@Row@bg{\appto\kvt@@row{\kvt@userowcolors}}
\eappto\kvt@@row{\noexpand\rowcolor{\expandonce\cmdkvt@Row@bg}}%
If a row alignment is specified, a default \multicolumn display is enabled for the row's cells.

\ifdef\cmdkvt@Row@align{\let\kvt@roukmulticolumn\@empty}
\edef\kvt@roukmulticolumn{%
\noexpand\multicolumn{1}{\expandonce\cmdkvt@Row@align}}%
The following defines a macro \kvt@cellfmtbuilder\langle cmd \rangle{\langle csname \rangle}. This macro defines the macro \langle cmd \rangle{\langle cell \rangle} to format the cell content, \langle cell \rangle, based on the column format \langle csname \rangle and the row formatting options. Through this “builder” macro, the row format options need only be considered once and the
column format options can then be included when the displayed columns are iterated over.

\ifcsvoid{cmdkvt@Row@format!}
\edef\kvt@@cellfmtbuilder##1##2{\noexpand\edef##1####1{\noexpand\kvt@expandonce@onearg\noexpand\kvt@@mkmulticolumn{\ifcsvoid{cmdkvt@Row@format*}{\@firstofone}{\noexpand\unexpanded{\csexpandonce{cmdkvt@Row@format*}}{\noexpand\csexpandonce{##2}{\ifdefvoid\cmdkvt@Row@format{\@firstofone}{\noexpand\unexpanded\expandonce\cmdkvt@Row@format}}{####1}}}}}}}
\edef\kvt@@cellfmtbuilder##1##2{\noexpand\edef##1####1{\noexpand\kvt@expandonce@onearg\noexpand\kvt@@mkmulticolumn{\ifcsvoid{cmdkvt@Row@format!}{\@firstofone}{\noexpand\unexpanded\expandonce\cmdkvt@Row@Row@format}&####1}}}\}
\do{\langle cname \rangle}

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
\kvt@def@atseconduse\kvt@@switchcol{\appto\kvt@@row{&}}
\def\do##1{\ifnumgreater\kvt@@span{0}{\advance\kvt@@span\m@ne\ifcsvoid{cmdKeyValTable@#2@##1}{}{\ifdefvoid\kvt@@curcgname{\kvt@error{Column '##1' nonempty inside a \string\multicolumn}{}}{\kvt@error{Column '##1' nonempty inside column group '\kvt@@curcgname'}{}}}}\kvt@@switchcol

First, 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 the column is simply ignored and otherwise an error is produced.

\ifnum\kvt@@span=0\relax
\ifcsvoid{cmdKeyValTable@#2@##1}{\let\kvt@@cell=\kvt@col@default@#2@##1}{\let\kvt@@cell=cmdKeyValTable@#2@##1}\unless the default cell value is used, first check for a multicolumn value. Default cell values should not need this. The check is done before the expansion code afterwards, in order for applying the expansion to the code in the cell value rather than to the multicolumn code.
Apply expansion control options, but only to manually supplied cell values, not to default values.

Separately also already create the content – with formatting unless the user explicitly requested no cell formatting.

Finally, append the cell to the row.

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

At the very end of the expansion text, put \post. 

\kvt@def@atseconduse\{\cmd\}{\code} defines the macro \cmd to expand to \code but only from its second use onwards. At its first use, \cmd only redefines itself to \code but does not do anything else.

\kvt@expandonce@onearg\{\cmd\}{\arg} macro expands to \arg if \cmd is empty and expands to an \expandonce on \cmd with \arg as argument otherwise. This macro is for an \edef context in which an empty \cmd should not leave any parentheses around the \arg.

Note that the alternative of avoiding the conditional (\ifdefequal) in the above code and using \@firstofone instead of \empty for a noop in \cmd does not work: Using \expandonce\{\cmd\}{\arg} would expand to \expandonce\{\empty\} and produces the error ‘Argument of \@firstofone has an extra ’. Using \expandonce\{\cmd\}{\arg}\} would expand to \expandonce\{\empty\}\} and, thus, prevent expansion of \arg.

\kvt@expandonce@onearg
\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{(content)}{(tname)} macro checks whether a cell's (content) in a table of type (tname) spans multiple columns in one of two ways:

1. (content) = \multicolumn{(n)}{(align)}{(content)} or
2. (content) = \kvt@@@colgroup{(cgname)}{(n)}{(align)}{(content)}

The first way corresponds to the case that a user of the package explicitly assigns a \multicolumn expression to a cell in a row. The second way is generated by the package when a user assigns a normal cell value to a column group key.

\newcommand\kvt@CheckMulticolumn[2]{%
For parsing (content), the macro uses \kvt@CheckMulticolumn@i and adds 5 \relax after (content) for the case that (content) is empty or too short.

\kvt@CheckMulticolumn@i{#2}#1%
\relax\relax\relax\relax\relax\kvt@@undefined}

\kvt@CheckMulticolumn@i

The \kvt@CheckMulticolumn@i{(tname)}{⟨c1⟩⋯⟨c5⟩}{⟨ign⟩}\@undefined macro checks (content) when split into ⟨c1⟩⋯⟨c5⟩ for one of the two multicolumn cases listed in the description of \kvt@CheckMulticolumn.

\def\kvt@CheckMulticolumn@i#1#2#3#4#5#6#7\kvt@@undefined{%
\ifdefmacro{#2}{%
First case: ⟨c1⟩=\multicolumn. In this case, we have ⟨c2⟩=(n), ⟨c3⟩=(align), and ⟨c4⟩=(content).

\ifdefvoid{\cmdkvt@Row@align}{}
\ifdefmacro{#5}{%
Second case: ⟨c1⟩=\kvt@@@colgroup. In this case, we have ⟨c3⟩=(n), ⟨c4⟩=(align), and ⟨c5⟩=(content). Moreover, ⟨c2⟩ holds ⟨cgname⟩.

\ifdefvoid{\cmdkvt@Row@align}{}
\else{%
If a row alignment is defined, it overrides the alignment of the column group:

\ifdefvoid{\cmdkvt@Row@align}{}
\expandafter\kvt@SetMulticolumn\expandafter{#4}{#3}{#5}%\n\let\kvt@@curcgname@empty\empty
}\fi{%}
\fi{%}
\fi{%
\else{%
The \kvt@@@colgroup macro is not used as an actual macro but only as an identifier for \kvt@CheckMulticolumn@i.
\newcommand\kvt@@@colgroup{kvt@@@colgroup}
The `\kvt@SetMulticolumn{⟨align⟩}{⟨n⟩}{⟨content⟩}` records that ⟨n⟩ cells, starting from the current cell, belong to a multicolumn cell with alignment ⟨align⟩ and the given ⟨content⟩.

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.

Next, unwrap the cell’s ⟨content⟩ to `\kvt@@cell` and record the `\kvt@@mkmulticolumn` for re-wrapping the content later, after all cell formatting has been applied.

Next, unwrap the cell’s ⟨content⟩ to `\kvt@@cell` and record the `\kvt@@mkmulticolumn` for re-wrapping the content later, after all cell formatting has been applied.

### 10.7.4 Table and Row Styles

The following are the user macros.

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

The `\kvtNewTableStyle{⟨name⟩}{⟨options⟩}` macro declares ⟨name⟩ as a table style and defines it to be equivalent to specifying ⟨options⟩ directly in the optional argument of a `\KeyValTable` environment or of a `\NewKeyValTable`. The macro fails if ⟨name⟩ is already declared as a table style. The `\kvtRenewTableStyle{⟨name⟩}{⟨options⟩}` macro re-defines an already existing table style with new ⟨options⟩.

The following are the internal macros that the style code shares.

The `\kvt@NewStyle{⟨type⟩}{⟨renewcmd⟩}{⟨name⟩}{⟨options⟩}` macro defines a new style, ⟨name⟩, for ⟨type⟩ (table or row) to correspond to ⟨options⟩. Analogously, `\kvt@RenewStyle{⟨type⟩}{⟨newcmd⟩}{⟨name⟩}{⟨options⟩}` macro re-defines a style.

The following are the user macros.
The `\kvt@UseRowStyles{styles}` and `\kvt@UseTableStyles{styles}` macros set the keys for the given, comma-separated list of `styles`.

```latex
\newcommand\kvt@UseRowStyles[1]{\kvt@UseStyles{row}{Row}{\kvt@NewRowStyle}{#1}}
\newcommand\kvt@UseTableStyles[1]{\kvt@UseStyles{table}{Table}{\kvt@NewTableStyle}{#1}}
```

The `\kvt@UseStyle{type}{fam}{newcmd}{style}` macro sets the keys for type `type` based on the `options` stored for the given `style`. The `fam` identifies the `xkeyval` family for `type` and `newcmd` is the macro for defining new `type` styles.

```latex
\newcommand\kvt@UseStyle[4]{%  
  \ifcsundef{kvt@@style@#4}{\kvt@error{A #1 style '#4' is not defined}{Use \string#4\space to define a new #1 style.}}{\kvt@setcskeys{kvt@@style@#4}{#2}}}
```

The `\kvt@UseStyles{type}{fam}{newcmd}{styles}` macro sets the `type` keys based on the `options` for all styles in the comma-separated list `styles`. The `fam` identifies the `xkeyval` family for `type` and `newcmd` is the macro for defining new `type` styles.

```latex
\newcommand\kvt@UseStyles[4]{%  
  \ifnumgreater{\XKV@depth}{1}{#3}{\kvt@xkv@savepreset{#1}{#2}{h}\kvt@xkv@savepreset{#1}{#2}{t}\kvt@xkv@restorepreset{#1}{#2}{h}\kvt@xkv@restorepreset{#1}{#2}{t}}}
```

We use `\kvt@xkv@disablepreset` to eliminate undesired effects that would otherwise be caused by preset values for keys. For an example of such side-effect, consider a style “vis” that is defined as “hidden=false”. Then, `\Row[bg=red,style=vis]{...}` causes a `\setkeys[kvt]{Row}{hidden=false}` to be processed inside the `\setkeys[kvt]{Row}{bg=red,style=vis}`, after the `bg=red` is processed. The former `\setkeys` would then again employ the presets for Row (e.g., from a `\kvtSet{Row/bg=blue}`) and undesirably overwrite the `bg=red`.

```latex
\kvt@xkv@disablepreset[2]{%  
  \ifnumgreater{\XKV@depth}{1}{#3}{\kvt@xkv@savepreset{#1}{#2}{h}\kvt@xkv@savepreset{#1}{#2}{t}\kvt@xkv@restorepreset{#1}{#2}{h}\kvt@xkv@restorepreset{#1}{#2}{t}}}
```

The auxiliary macro `\kvt@xkv@savepreset{prefix}{family}{h/t}` saves and unsets the preset keys (head keys for `h/t=1` and tail keys otherwise) for `family`. The macro `\kvt@xkv@restorepreset{prefix}{family}{h/t}` restores the preset keys saved via `\kvt@xkv@savepreset`.

```latex
\kvt@xkv@savepreset[3]{%  
  \csletcs{kvt@@saved@preset#3}{XKV@#1@#2@preset#3}}
```

```latex
\kvt@xkv@restorepreset[3]{%  
  \csletcs{kvt@@saved@preset#3}{XKV@#1@#2@preset#3}}
```
10.8 Collecting Key-Value Table Content

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⟩.

\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⟩.

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

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

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.

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

1. protects ordinarily ‘protec’ted parts via \kvtprotected@write;

2. protects table macros – like \tkvtRow, 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;
3. protects table counters like \kvtRow by adapting the counter-formatting macros to treat table counters differently from other counters.

\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.

\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\.

\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\.

\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
The macro records the \textit{macro-list} by appending the csnames of its elements to \texttt{\kvt@numberformatters}. The actual expansion control for the macros in \textit{macro-list} is performed by \texttt{\kvt@writeprotect@fmt}.

\begin{verbatim}
\newcommand{\kvtDeclareCtrFormatters}[1]{%  
  \def{do##1}{\listadd{\kvt@@numberformatters}{\expandafter{\@gobble}{\string##1}}}%  \docsvlist{#1}}
\end{verbatim}

Initially empty \texttt{etoolbox} list of counter-formatting macros.

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

\texttt{\kvt@RecordedRow} appends a \Row with \texttt{options} and \texttt{content} to a global macro for \texttt{cname}.

\begin{verbatim}
\newcommand{\ShowCollectedTable}[2][]{%  \ifcsvoid{\kvt@@tnameof@#2}{\kvt@error{No row collection with name '#2' defined}  \kvt@warn{No row data available for name '#2.}}{\ifcsvoid{\kvt@@rowsof@#2}{\kvt@warn{No row data available for name '#2.}}{\newline A LaTeX rerun might be needed}}}\end{verbatim}

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.

\begin{verbatim}
\kvtDeclareTableMacros{\thekvtRow, \thekvtTypeRow, \thekvtTotalRow}\end{verbatim}

\begin{verbatim}
\kvtDeclareTableCounters{kvtRow, kvtTypeRow, kvtTotalRow}\end{verbatim}

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

\begin{verbatim}
\kvtDeclareCtrFormatters{\arabic, \alph, \Alph, \roman, \Roman, \fnsymbol}\end{verbatim}

The \texttt{\kvt@coldefault{(tname)}{(cname)}} macro expands to \texttt{\texttt{\texttt{(cname)}={\texttt{(default)}},}}, where \texttt{(default)} is the default value of column \texttt{(cname)} in table type \texttt{(tname)}. If \texttt{(default)} is empty, then the macro expands to the empty string. The \texttt{\kvt@coldefaults@i{(tname)}} macro expands to the comma-separated list of the \texttt{\kvt@coldefault} for all \textit{displayed} columns of table type \texttt{(tname)}. Finally, the \texttt{\kvt@coldefaults{(cname)}} macro expands to \texttt{\kvt@coldefaults} for the table type assigned to \texttt{(cname)} via \texttt{\NewCollectedTable}.

\begin{verbatim}
\newcommand{\kvt@coldefaults}[1]{%  \forlistcsloop{\kvt@coldefault}{kvt@displaycols@#1}{#1}}\end{verbatim}

\texttt{\kvt@coldefault} expands to \texttt{\kvt@coldefault{(tname)}{(cname)}} which is a list of \texttt{(cname)} assignments, and \texttt{\kvt@coldefaults{(cname)}} expands to \texttt{\kvt@coldefaults} for the table type assigned to \texttt{(cname)} via \texttt{\NewCollectedTable}.
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\}\}

\subsection{Table Content from Files}

The \ShowKeyValTableFile\[\{\options\}\]\{\tname\}\{\filename\}\ loads the content of the file with name \filename and places it inside the body of a KeyValTable environment of type \tname with the given \options. That is, the filename should contain the rows of the table.

\newcommand\ShowKeyValTableFile\[3\]\{\begin{KeyValTable}\[#1\]\{\#2\}\csuse{kvt@rows@#2}\end{KeyValTable}\csdef{kvt@rows@#2}{\}\}

\subsection{Legacy Variant}

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\}\end{KeyValTable}\csuse{kvt@rows@#2}\csdef{kvt@rows@#2}{\}\}

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\[1\]\{\begin{group}\kvt@row\{\tname\}\{\#1\}\end{group}\}

\endinput
The `KeyValTableContent` 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⟩}.

\begin{verbatim}
newenvironment{KeyValTableContent}{% \def\Row{\AddKeyValRow{#1}}}{}%
\end{verbatim}

10.9 Package Options

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

\begin{verbatim}
\define@cmdkey[kvt]{PackageOptions}[kvt@@pkg@]{compat}{}
\end{verbatim}

Next, set default package options and process them.

\begin{verbatim}
\ExecuteOptionsX[kvt]<PackageOptions>{compat=2.0,}
\ProcessOptionsX[kvt]<PackageOptions>\relax
\end{verbatim}

10.10 Compatibility

\begin{verbatim}
{kvt@NewCompat}
\end{verbatim}

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

v0.1
General: Initial version ............ 1

v0.2
\NewKeyValTable: Added table-type options .......... 32
\kvtLabel: Added macro for row labeling ............. 41
General: Added “shape” table option ............. 28

v0.3
\kvt@StartTabularlike: Added showhead option ......... 47
\kvtLabel: Robustified for use with, e.g., cleveref .... 41
\kvtStrutted: Fix for cells with vertical material ........ 28

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

v1.0
\NewKeyValTable: Added optional headers argument .......... 32
Added zero-width column for \multicolumn ............. 33
\kvt@AddKeyValRow: Added \{options\} ............. 50
\kvt@AddKeyValRow@ii: Added \multicolumn support .......... 51
\kvt@StartTabularlike: Added width option .......... 47
Implemented showrules option .......... 45
General: Enabled default “true” for “hidden” .......... 30

v2.0
\CollectRow: Added the macro ................ 56
\NewCollectedTable: Added the macro ............. 56
\NewKeyValTable: Changed headers argument to layout argument .......... 32
\ShowCollectedTable: Added the macro ............. 58
\ShowKeyValTableFile: Added the macro ............. 59
\kvtNewRowStyle: Added the macro ............. 54
\kvtRenewRowStyle: Added the macro ............. 54
\kvtStrutted: Added optional argument ............. 28
General: added package option “compat” ............. 60
added row option “style” .......... 31
added row option “uncounted” .......... 31
added row options “expand” and “expandonce” .......... 31
added row options “nobg” and “norowbg” .......... 31
Added table options “caption” and “label” .......... 28

v2.1
\NewKeyValTable: Removed zero-width column again ................ 33
\kvt@StartTabularlike: Added valign and halign options .......... 47
General: Added “valign” and “halign” table options .......... 28
added abbreviation option “norules” .......... 29
added row options “format”, “format”, “align”, and “headlike” .......... 31

v2.2
\kvtUseTableStyles: Added the macro ................ 55
\kvtNewTableStyle: Added the macro ............. 54
\kvtRenewTableStyle: Added the macro ............. 54
General: Added “style” table option ............. 28
added option “backend” ............. 28
added option “captionpos” .......... 28
added options “caption/lot” and “caption/alt” .......... 29
added table options “resume” and “resume*” .......... 29
Index

Symbols
[@input 870
[@auxout 813
[@empty 350, 513, 514, 524, 525, 546, 547, 548, 549, 661, 676, 734, 746
[@firstofone 38, 162, 683, 686, 718
[@firstoftwo 248, 508
[@gobble 12, 836
[@ifnextchar 184, 655
[@ifstar 8
[@mkpream 509
[@ne 356, 363, 384, 669, 832
[@nil 205, 247, 600, 601
[@secondoftwo 249, 507
[@undefined 202, 208, 258, 261, 348, 385
[@ 23, 534, 540
A
above (option-key) 13
AddKeyValRow 5, 879, 885
addtocounter 406, 736, 737, 738
advance 356, 669, 698, 757
align (option-key) 3, 13, 16, 17
Alph 841
alph 841
appto 45, 337, 340, 341, 353, 366, 518, 519, 520, 669, 671, 673, 695, 723, 727
arabic 841
around (option-key) 13
AtEndOfPackage 157
B
backend (option-key) 9
baselineskip 597
begin 861, 870, 875
begingroup 256, 333, 347, 506, 569, 809, 881
below (option-key) 13
bg (option-key) 13
bgroup 414
C
caption 544, 595, 598
caption (option-key) 18
caption/alt (option-key) 18
caption/lot (option-key) 18
captionpos (option-key) 19
cmdkvt@Row@above 666, 668
cmdkvt@Row@align 675, 678, 750, 753
cmdkvt@Row@below 728, 730
cmdkvt@Row@bg 672, 674
cmdkvt@Row@format 686, 687
cmdkvt@Table@captionpos 528, 538, 545
cmdkvt@Table@align 558, 559, 571, 572, 579, 580
cmdkvt@Table@headalign 142, 241, 243
cmdkvt@Table@headbg 140, 236, 368
cmdkvt@Table@headformat 141, 242, 244, 382
cmdkvt@Table@rowbg 568
cmdkvt@Table@shape 418, 420
cmdkvt@Table@valign 553, 554, 578
cmdkvt@Table@width 575, 576
cmdkvt@TableEnv@caption 523, 531
cmdkvt@TableEnv@label 532, 533
CollectRow 5, 804
counters:
kvtRow 7
cvtTotalRow 7
cvtTypeRow 7
csappto 205, 225, 367
csdef 59, 61, 63, 74, 78, 82, 111, 128, 130, 192, 193, 194, 195, 196, 197, 198, 199, 204, 231, 232, 304, 331, 332, 396, 439, 630, 766, 773, 824, 878
cseappto 223, 226, 338
csdef 341, 632, 637, 640
csexpandonce 223, 226, 283, 339, 381, 383, 528, 530, 538, 539, 581, 684, 685, 692
csgappto 849
csgdef 117, 119, 121, 801

62
D
\DeclareListParser .... 11, 21
default (option-key) ....... 3
\define@boolkey .... 56, 57, 65, 67, 69, 89, 94, 107, 137, 145, 151, 152, 153, 154, 245, 246, 247, 429, 431, 433
\define@choicekey ... 58, 60, 62
\define@cmdkey 51, 52, 53, 54, 55, 85, 86, 87, 88, 116, 118, 120, 123, 125, 132, 133, 134, 135, 136, 146, 147, 229, 276, 435, 886
\define@key 64, 73, 77, 103, 104, 105, 106, 113, 127, 129, 148, 150, 230, 436
\do 201, 258, 335, 348, 354, 367, 370, 455, 459, 696, 818, 832, 836
\docsvlist ... 457, 461, 833, 837
\dolistcsloop .... 364, 725
\dolistloop .... 819
E
\eappto 275, 379, 551, 667, 674, 729
\egroup ................. 414
\else .................... 248, 747
\end ..................... 863, 870, 877
\endsname 12, 30, 37, 50, 327, 422, 443, 463, 466, 570, 633, 635, 638, 641, 646, 826, 827, 865
\endfirsthead .......... 585
\endfoot ................... 593
\endgroup 260, 344, 370, 510, 569, 812, 882
\endhead .................. 588
\endlastfoot ............ 591
\endlongtable ....... 485, 503
\endtabular ............... 471, 497
environments:
KeyValTable ........ 4, 412
KeyValTableContent : 6, 884
\etb@listitem ........... 13, 20
\ExecuteOptionsX ......... 887
expand (option-key) ....... 23
\expandafter .... 11, 12, 18, 28, 30, 35, 37, 39, 124, 126, 143, 212, 248, 249, 260, 311, 313, 325, 327, 344, 357, 370, 373, 414, 421, 422, 423, 442, 445, 466, 478, 479, 491, 510, 568, 570, 633, 635, 638, 641, 644, 645, 711, 713, 720, 721, 723, 752, 753, 826, 836, 864, 865, 866, 867
\expandonce .... 140, 141, 142, 242, 243, 244, 280, 282, 369, 531, 533, 575, 576, 582, 583, 586, 587, 589, 592, 608, 614, 674, 678, 687, 720, 730, 734, 882
expandonce (option-key) ...... 23
\fi .................... 39, 249, 312, 320, 357, 754
\fnsymbol ................. 841
\forcsvlist .... 115, 785, 829
\forlistcsloop ... 219, 323, 369
\forlistloop ............ 820
format (option-key) ...... 3, 13, 17
format* (option-key) ...... 13
G
\global .................. 563, 669
H
halign (option-key) ........ 9
head (option-key) ...... 3, 17
headalign (option-key) ...... 10
headbg (option-key) ...... 10
headformat (option-key) ...... 10
headlike (option-key) ...... 13
hidden (option-key) ...... 3, 13
I
\ifbool 70, 90, 95, 138, 416, 448, 449, 450, 451, 452, 453, 515, 550, 564, 660, 670, 712, 714
\ifcase \ifcsdef \ifcsstring \ifcsundef \ifcsvoid 
\ifdefequal \ifdefmacro \ifdefstring \ifdefvoid 
\ifdimcomp \IfFileExists \ifhmode \ifinlist \ifinlistcs 
\ifnum \ifnumgreater \ifnumodd \ifstrempty \ifstrequal \ifundef \ifx 
\kvt@@@colgroup \kvt@@bgcolor@even \kvt@@bgcolor@odd 
\kvt@@bodyrow \kvt@@caption@alt \kvt@@caption@footalt \kvt@@caption@footmain 
\kvt@@caption@headalt \kvt@@caption@headmain \kvt@@caption@main 
\kvt@@cell \kvt@@cellfmtbuilder \kvt@@coldo \kvt@@colgrp \kvt@@colgrp@first \kvt@@colgrp@n 
\kvt@@colreg \kvt@@column \kvt@@curcgname \kvt@@curcolformat \kvt@@curgrp \kvt@@do 
\kvt@@endhook \kvt@@fmtcell \kvt@@formatter \kvt@@hdcell \kvt@@lastenvopt \kvt@@lasthd 
\kvt@@mkmulticolumn \kvt@@numberformatters \kvt@@opts \kvt@@parseheadrows \kvt@@presetqueue 
\kvt@@psvdo \kvt@@result \kvt@@row \kvt@@rowcountlast \kvt@@rowmkmulticolumn \kvt@@rule

K
KeyValTable (environment) \  4, 412
KeyValTableContent (environment) \  6, 884
\kvt@@@caption@main \kvt@@caption@main \kvt@@caption@main 
524, 527, 546, 548 \kvt@@cell \kvt@@cell \kvt@@cell 
709, 710, 711, 713, 715, 722, 758 \kvt@@cellfmtbuilder \kvt@@cellfmtbuilder 
680, 689, 717, 726 \kvt@@colo\ldo \kvt@@colo\ldo \kvt@@colo\ldo 
309, 323 \kvt@@colgrp \kvt@@colgrp \kvt@@colgrp 
117, 119, 121, 272, 316 \kvt@@colgrp@first \kvt@@colgrp@first \kvt@@colgrp@first 
277, 280, 307, 314 \kvt@@colgrp@n \kvt@@colgrp@n \kvt@@colgrp@n 
282, 306, 321 \kvt@@colreg \kvt@@colreg \kvt@@colreg 
387, 397 \kvt@@column \kvt@@column \kvt@@column 
110, 111, 210, 211, 212, 215, 234 \kvt@@curcgname \kvt@@curcgname \kvt@@curcgname 
700, 704, 746, 749 \kvt@@curco\ldef \kvt@@curco\ldef \kvt@@curco\ldef 
707, 748 \kvt@@curgr\gr \kvt@@curgr\gr \kvt@@curgr\gr 
318 \kvt@@curhd \kvt@@curhd \kvt@@curhd 
352, 354, 355, 358, 359, 361, 363 \kvt@@do \kvt@@do \kvt@@do 
569, 594 \kvt@@endhook \kvt@@endhook \kvt@@endhook 
512, 513, 519, 520 \kvt@@fmtcell \kvt@@fmtcell \kvt@@fmtcell 
720, 724 \kvt@@formatter \kvt@@formatter \kvt@@formatter 
717, 718, 721 \kvt@@hdcell \kvt@@hdcell \kvt@@hdcell 
128, 130, 398 \kvt@@lastenvopt \kvt@@lastenvopt \kvt@@lastenvopt 
96, 100, 417 \kvt@@lasthd \kvt@@lasthd \kvt@@lasthd 
352, 355, 363, 378, 381, 383, 384 \kvt@@mkmulticolumn \kvt@@mkmulticolumn \kvt@@mkmulticolumn 
682, 691, 706, 759 \kvt@@numberformatters \kvt@@numberformatters \kvt@@numberformatters 
820, 835, 838 \kvt@@opts \kvt@@opts \kvt@@opts 
139, 143 \kvt@@parseheadrows \kvt@@parseheadrows \kvt@@parseheadrows 
334, 337, 340, 341, 344 \kvt@@pkg@compat \kvt@@pkg@compat \kvt@@pkg@compat 
892 \kvt@@prehook \kvt@@prehook \kvt@@prehook 
514, 518, 551, 583, 587 \kvt@@presetqueue \kvt@@presetqueue \kvt@@presetqueue 
42, 43, 45 \kvt@@psvdo \kvt@@psvdo \kvt@@psvdo 
294, 305 \kvt@@result \kvt@@result \kvt@@result 
257, 260, 275 \kvt@@row \kvt@@row \kvt@@row 
414, 661, 665, 667, 669, 671, 673, 674, 695, 723, 727, 729, 882 \kvt@@rowcountlast \kvt@@rowcountlast \kvt@@rowcountlast 
90 \kvt@@row\ldoi\ldo \kvt@@row\ldoi\ldo \kvt@@row\ldoi\ldo 
676, 677, 706 \kvt@@rowmkmulticolumn \kvt@@rowmkmulticolumn \kvt@@rowmkmulticolumn 
676, 677, 706 \kvt@@rule \kvt@@rule \kvt@@rule 
516, 517, 518, 519, 551

64
\kvt@RegisterShape 607, 627, 895, 896, 899, 900
\kvt@RegisterTableStyle ... 761, 763, 764
\kvt@RegisterRowcolorcmdornot 371, 605, 606
\kvt@RegisterRowcolorornot 236, 368, 371
\kvt@RegisterRowkeys ........... 26, 32, 41, 66, 68, 71, 72, 101, 221, 253, 273, 399, 424, 659, 810
\kvt@RegisterRowkeys@nopresets ... 31, 143, 149
\kvt@RegisterTableMultiColumn ... 745, 751, 753, 756
\kvt@RegisterTableOptions ....... 415, 421
\kvt@RegisterTableOptions@i ....... 422, 423
\kvt@RegisterRowcolors .......... 568, 599
\kvt@RegisterRowcolors@i ........ 600, 601
\kvt@RegisterStartTabularlike ... 511, 631, 639, 642
\kvt@RegisterStepCounters ... 671, 735
\kvt@RegisterTableOf .......... 860
\kvt@RegisterTableOfCname ... 858, 859, 860
\kvt@RegisterTableOfCname@i ... 860
\kvt@RegisterRowcolors .......... 604, 673
\kvt@RegisterTableRowStyle ...... 150, 778
\kvt@RegisterTableRowStyle ...... 778, 785
\kvt@RegisterTableRowStyles .... 775, 777, 783
\kvt@RegisterTableRowStyles .... 64, 774
\kvt@RegisterWarn 14, 24, 80, 555, 560, 855
\kvt@RegisterWriteProtect@fmt . 820, 822
\kvt@RegisterXKV@disablepreset . 32, 278, 784, 786
\kvt@RegisterXKV@restorepreset 792, 793, 794
\kvt@RegisterXKV@savepreset .. 789, 790, 794
\kvt@RegisterDeclareCtrFormatters 8, 834, 841
\kvt@RegisterDeclareTableCounters 8, 831, 840
\kvt@RegisterDeclareTableMacros . 8, 828, 839
\kvt@Label .................. 7, 404
\kvt@NewRowStyle ........... 14, 700
\kvt@NewTableStyle .......... 10, 762
\kvt@RenewRowStyle ........... 14, 700
\kvt@RenewTableStyle ....... 12, 762
\kvt@Row ...................... 400
\kvt@Row(counter) ............ 7
\kvt@Set .................... 15, 40, 157
\kvt@Strutted ................ 12, 38, 168, 180
\kvt@TableOpt ................ 50
\kvt@TotalRow ............... 402
\kvt@TotalRow(counter) ...... 7
\kvt@TypeRow ................ 401
\kvt@TypeRow(counter) ...... 7

L
\label ..................... 409, 533
label (option-key) ........ 18
\let 307, 350, 363, 513, 514, 524, 525, 546, 547, 548, 549, 661, 676, 706, 713, 718, 746
\letcs ... 354, 707, 709, 710, 748
\linewidth .................. 165
\listadd .... 200, 612, 619, 829
\listcsadd .... 220, 227, 287
\listeadd ........ 835

M
\m@ne .................... 698, 757
\metatbl@branch ... 507, 508, 510
\metatbl@envname 438, 443, 445, 446
\metatbl@boolprop 425, 426, 428, 430, 432, 434, 444
\metatbl@csnamearg 456, 460, 464
\metatbl@ifhasXcolumns 505, 644
\metatbl@setprop 435, 436, 441
\metatbl@AtEnd .......... 462, 512
\metatbl@CanHAlign ....... 447, 558
\metatbl@CanVAlign ....... 447, 553
\metatbl@HasCaption ....... 447, 526
\metatbl@HasWidth ........ 447, 573
\metatbl@isLong ......... 447, 584, 590
\metatbl@IsTabu .. 447, 572, 574, 580
\metatbl@Registered .... 447, 623
\metatbl@RegisterEnv . 437, 467, 473, 481, 487, 493, 499, 625
\metatbl@Require .... 454, 894, 898
\metatbl@UsePackage .... 454
\metatbl@UsePackage .... 454
\multicolumn . 243, 379, 678, 702, 744, 759
\NC@find ................. 508
\NC@rewriteDX ............. 508
\NewCollectedTable . 4, 799, 807, 853
\newcount ................. 374, 375
\newcounter .............. 400, 401, 402, 411
\NewKeyValTable 2, 183, 266, 270, 298, 319, 362, 390, 394
\newrobustcmd ............. 437, 505
\noalign .................. 414, 520, 597, 667, 669, 671, 729
\nobg (option-key) ........ 10
\norowbg (option-key) ...... 10
\norules (option-key) ...... 9
\numexpr .................. 321, 342

O

option-keys:
above .................. 13
align .................. 3, 13, 16, 17
around .................. 13
backend ................. 9
below .................. 13
bg ..................... 13
caption ................ 18
caption/alt ............ 18
caption/lot ............ 18
captionpos ............ 19
default ............... 3
expand ................ 23
expandonce ............ 23
format ................ 3, 13, 17
format* ................ 13
halign ................ 9
head .................. 3, 17
headalign ............. 10
headbg ............... 10
headformat ........... 10
headlike ............. 13
hidden ................ 3, 13
label .................. 18
nobg .................. 10
norowbg ............... 10
norules ............... 9
resume ................ 7
resume* ............... 12
rowbg ............... 10
shape ................ 9
showhead .............. 9
showrules ............ 9
span .................. 17
style ................ 10, 14
uncounted ............ 10
valign ................ 9
width ................ 9
\or .................. 311, 315

P
\PackageError ........... 24
\PackageWarning ........ 25
\parbox .................. 597
\PassOptionsToPackage ... 4
\patchcmd ............... 12
\presetkeys ............ 45, 233
\preto 471, 478, 479, 485, 491, 497, 503
\ProcessOptionsX ...... 890
\protected@edef ........ 247, 307, 321, 342, 563, 694, 741, 757, 890
\protected@write ....... 471, 478, 479, 485, 491, 497, 503
\providebool ........... 445

R
\refstepcounter .......... 407
\relax 247, 307, 321, 342, 563, 694, 741, 757, 890
\renewcommand .......... 903
\RequirePackage 1, 2, 3, 5, 6, 460
\resume (option-key) .... 7
\resume* (option-key) ... 12
\Roman ................ 841
\roman ................ 841
\Row .................. 4, 413, 849, 885
\rowbg (option-key) ..... 10
\rowcolor ............. 371, 373, 674

S
\setbool ................. 446
\setcounter 403, 405, 565, 566, 567
\setkeys \showCollectedTable \showKeyValTable \showKeyValTableFile \showrules \space \span \string \strut \style \vskip \vspace

T \tabularnewline \the \toks@ \trim@post@space@noexp

U \uncounted \edef \unexpanded \usepackage

V \valign \value \vskip \vspace

W \width \vskip

X \XKV@depth \XLT@i@TX@endtabularx \XLT@ii@TX@endtabularx

Z \z@