Package 'basictabler'

May 7, 2026

Type Package

Title Construct Rich Tables for Output to 'HTML'/'Excel'

Version 1.0.4

Description Easily create tables from data frames/matrices. Create/manipulate tables row-by-row, column-by-column or cell-by-cell. Use common formatting/styling to output rich tables as 'HTML', 'HTML widgets' or to 'Excel'.

Depends R (>= 3.3.0)

Imports R6 (>= 2.2.0), dplyr (>= 0.5.0), htmltools(>= 0.3.5), htmlwidgets (>= 0.8)

Suggests jsonlite (>= 1.1), lubridate (>= 1.5.0), listviewer (>= 1.4.0), openxlsx (>= 4.0.17), flextable (>= 0.6.6), officer (>= 0.3.18), shiny, knitr, rmarkdown, testthat

License GPL-3

Encoding UTF-8

LazyData true

URL http://www.basictabler.org.uk/, https://github.com/cbailiss/basictabler

BugReports https://github.com/cbailiss/basictabler/issues

VignetteBuilder knitr

RoxygenNote 7.3.2

NeedsCompilation no

Author Christopher Bailiss [aut, cre]

Maintainer Christopher Bailiss <cbailiss@gmail.com>

Repository CRAN

Date/Publication 2025-04-21 21:40:02 UTC

Contents

`BasicTable` *R6 class that defines a basic table.*

Description

The 'BasicTable' class represents a table with styling and formatting that can be rendered to multiple output formats.

Format

`R6Class` object.

Active bindings

`argumentCheckMode` The level of argument checking to perform. One of "auto", "none", "minimal", "basic", "balanced" (default) or "full".

`compatibility` A list containing compatibility options to force legacy behaviours. See the NEWS file for details.

`traceEnabled` A logical value indicating whether actions are logged to a trace file.

`cells` A 'TableCells' object containing all of the cells in the body of the table.

`allCells` A list of all of the cells in the table, where each element in the list is a 'TableCell' object.

`mergedCells` A 'TableCellRanges' object describing the merged cells.

`rowCount` The number of rows in the table.

`columnCount` The number of columns in the table.

`asCharacter` The plain-text representation of the table.

`theme` The name of the theme used to style the table. If setting this property, either a theme name can be used, or a list can be used (which specifies a simple theme) or a 'TableStyles' object can be used. See the "Styling" vignette for details and examples.

`styles` A 'TableStyles' object containing the styles used to theme the table.

`allowExternalStyles` Default 'FALSE', which means the 'TableStyles' object checks that style names specified for styling the different parts of the table must exist in the styles collection. If they do not an error will occur. Specify 'TRUE' to disable this check, e.g. if the style definitions are not managed by 'basictabler' but instead in an external system.

`allTimings` The time taken for various activities related to constructing the table.

`significantTimings` The time taken for various activities related to constructing the table, where the elapsed time > 0.1 seconds.

Methods

Public methods:

- `BasicTable$new()`
- `BasicTable$getNextInstanceId()`
- `BasicTable$addData()`
- `BasicTable$addMatrix()`
- `BasicTable$mergeCells()`
- `BasicTable$unmergeCells()`
- `BasicTable$applyCellMerges()`
- `BasicTable$formatValue()`
- `BasicTable$addStyle()`
- `BasicTable$createInlineStyle()`
- `BasicTable$setStyling()`
- `BasicTable$mapStyling()`
- `BasicTable$resetCells()`
- `BasicTable$getCells()`
- `BasicTable$findCells()`
- `BasicTable$print()`
- `BasicTable$asMatrix()`
- `BasicTable$asDataFrame()`
- `BasicTable$getCss()`
- `BasicTable$getHtml()`
- `BasicTable$saveHtml()`
- `BasicTable$renderTable()`
- `BasicTable$writeToExcelWorksheet()`
- `BasicTable$asFlexTable()`
- `BasicTable$trace()`
- `BasicTable$asList()`
- `BasicTable$asJSON()`
- `BasicTable$viewJSON()`
- `BasicTable$clone()`

Method `new()`**:** Create a new 'BasicTable' object.

Usage:

```
BasicTable$new(
  argumentCheckMode = "auto",
  theme = NULL,
  replaceExistingStyles = FALSE,
```

```
  tableStyle = NULL,
  headingStyle = NULL,
  cellStyle = NULL,
  totalStyle = NULL,
  compatibility = NULL,
  traceEnabled = FALSE,
  traceFile = NULL
)
```

Arguments:

`argumentCheckMode` The level of argument checking to perform. Must be one of "auto", "none", "minimal", "basic", "balanced" (default) or "full".

`theme` A theme to use to style the table. Either:
(1) The name of a built in theme, or
(2) A list of simple style settings, or
(3) A 'TableStyles' object containing a full set of styles.
See the "Styling" vignette for many examples.

`replaceExistingStyles` Default 'FALSE' to retain existing styles in the styles collection and add specified styles as new custom styles. Specify 'TRUE' to update the definitions of existing styles.

`tableStyle` Styling to apply to the table. Either:
(1) The name of a built in style, or
(2) A list of CSS style declarations, e.g.
'list("font-weight"="bold", "color"="#0000FF")', or
(3) A 'TableStyle' object.

`headingStyle` Styling to apply to the headings. See the 'tableStyle' argument for details.

`cellStyle` Styling to apply to the normal cells. See the 'tableStyle' argument for details.

`totalStyle` Styling to apply to the total cells. See the 'tableStyle' argument for details.

`compatibility` A list containing compatibility options to force legacy behaviours. See the NEWS file for details.

`traceEnabled` Default 'FALSE'. Specify 'TRUE' to generate a trace for debugging purposes.

`traceFile` If tracing is enabled, the location to generate the trace file.

Returns: No return value.

Method `getNextInstanceId()`**:** Get the next unique object instance identifier.

Usage:

```
BasicTable$getNextInstanceId()
```

Details: R6 classes cannot be easily compared to check if two variables are both referring to the same object instance. Instance ids are a mechanism to work around this problem. Each cell is assigned an instance id during object creation, which enables reliable reference comparisons.

Returns: An integer instance id.

Method `addData()`**:** Populate the table from a data frame, specifying headers and value formatting.

Usage:

```
BasicTable$addData(
  dataFrame = NULL,
  columnNamesAsColumnHeaders = TRUE,
  explicitColumnHeaders = NULL,
  rowNamesAsRowHeaders = FALSE,
  firstColumnAsRowHeaders = FALSE,
  explicitRowHeaders = NULL,
  numberOfColumnsAsRowHeaders = 0,
  columnFormats = NULL,
  fmtFuncArgs = NULL,
  columnCellTypes = NULL,
  baseStyleNames = NULL
)
```

Arguments:

`dataFrame` The data frame to generate the table from.

`columnNamesAsColumnHeaders` 'TRUE' to use the data frame column names as column headings in the table. Default value 'TRUE.'

`explicitColumnHeaders` A character vector of column names to use as column headings in the table.

`rowNamesAsRowHeaders` 'TRUE' to use the data frame row names as row headings in the table. Default value 'FALSE.'

`firstColumnAsRowHeaders` 'TRUE' to use the first column in the data frame as row headings in the table. Default value 'FALSE.'

`explicitRowHeaders` A character vector of row names to use as row headings in the table.

`numberOfColumnsAsRowHeaders` The number of columns to be set as row headers.

`columnFormats` A list that is the same length as the number of columns in the data frame, where each list element specifies how to format the values. Each list element can be either a character format string to be used with 'sprintf()', a list of arguments to be used with 'base::format()' or a custom R function which will be invoked once per value to be formatted.

`fmtFuncArgs` A list that is the same length as the number of columns in the data frame, where each list element specifies a list of arguments to pass to custom R format functions.

`columnCellTypes` A vector that is the same length as the number of columns in the data frame, where each element is one of the following values that specifies the type of cell: root, rowHeader, columnHeader, cell, total. The cellType controls the default styling that is applied to the cell. Typically only rowHeader, cell or total would be used.

`baseStyleNames` A character vector of style names (from the table theme) used to style the column values.

Returns: No return value.

Method `addMatrix()`**:** Populate the table from a matrix, specifying headers and value formatting.

Usage:

```
BasicTable$addMatrix(
  matrix = NULL,
  columnNamesAsColumnHeaders = TRUE,
```

```
  explicitColumnHeaders = NULL,
  rowNamesAsRowHeaders = FALSE,
  explicitRowHeaders = NULL,
  columnFormats = NULL,
  baseStyleNames = NULL,
  fmtFuncArgs = NULL
)
```

Arguments:

`matrix` The matrix to generate the table from.

`columnNamesAsColumnHeaders` 'TRUE' to use the matrix column names as column headings in the table. Default value 'TRUE.'

`explicitColumnHeaders` A character vector of column names to use as column headings in the table.

`rowNamesAsRowHeaders` 'TRUE' to use the matrix row names as row headings in the table. Default value 'FALSE.'

`explicitRowHeaders` A character vector of row names to use as row headings in the table.

`columnFormats` A list that is the same length as the number of columns in the matrix, where each list element specifies how to format the values. Each list element can be either a character format string to be used with 'sprintf()', a list of arguments to be used with 'base::format()' or a custom R function which will be invoked once per value to be formatted.

`baseStyleNames` A character vector of style names (from the table theme) used to style the column values.

`fmtFuncArgs` A list that is the same length as the number of columns in the data frame, where each list element specifies a list of arguments to pass to custom R format functions.

`firstColumnAsRowHeaders` 'TRUE' to use the first column in the matrix as row headings in the table. Default value 'FALSE.'

Returns: No return value.

Method `mergeCells()`**:** Merge table cells by specifying either:
The top left cell (rFrom, cFrom) and the merged cell size (rSpan, cSpan) or, The top left cell (rFrom, cFrom) and bottom-right cell (rTo, cTo), or The ranges of rows/columns as vectors using rowNumbers and columnNumbers.

Usage:

```
BasicTable$mergeCells(
  rFrom = NULL,
  cFrom = NULL,
  rSpan = NULL,
  cSpan = NULL,
  rTo = NULL,
  cTo = NULL,
  rowNumbers = NULL,
  columnNumbers = NULL
)
```

Arguments:

`rFrom` The row-number of the top-left cell being merged.

`cFrom` The column number of the top-left cell being merged.

`rSpan` The number of rows that the merged cell spans.

`cSpan` The number of columns that the merged cell spans.

`rTo` The row-number of the bottom-right cell being merged.

`cTo` The column-number of the bottom-right cell being merged.

`rowNumbers` A vector specifying the row numbers of the cells to be merged.

`columnNumbers` A vector specifying the columns numbers of the cells to be merged.

Returns: No return value.

Method `unmergeCells()`**:** Unmerge a set of merged cells by specifying any cell within the set of merged cells.

Usage:

```
BasicTable$unmergeCells(r = NULL, c = NULL, errorIfNotFound = TRUE)
```

Arguments:

`r` The row number of any cell within the merged cell.

`c` The column number of any cell within the merged cell.

`errorIfNotFound` 'TRUE' to ignore any attempt to unmerge a cell that is not merged. Default value 'TRUE.'

Returns: A new 'TableCell' object.

Method `applyCellMerges()`**:** Internal method that sets the 'isMerged' and 'mergeIndex' properties on each cell based on the cell merges that have been specified.

Usage:

```
BasicTable$applyCellMerges()
```

Returns: No return value.

Method `formatValue()`**:** Format a value using a variety of different methods.

Usage:

```
BasicTable$formatValue(value = NULL, format = NULL, fmtFuncArgs = NULL)
```

Arguments:

`value` The value to format.

`format` Either a character format string to be used with 'sprintf()', a list of arguments to be used with 'base::format()' or a custom R function which will be invoked once per value to be formatted.

`fmtFuncArgs` If 'format' is a custom R function, then 'fmtFuncArgs' specifies any additional arguments (in the form of a list) that will be passed to the custom function.

Returns: The formatted value if 'format' is specified, otherwise the 'value' converted to a character value.

Method `addStyle()`**:** Add a new named style to the table.

Usage:

```
BasicTable$addStyle(styleName = NULL, declarations = NULL)
```

Arguments:

`styleName` The name of the new style.

`declarations` CSS style declarations in the form of a list, e.g. 'list("font-weight"="bold", "color"="#0000FF")'

Returns: The newly created 'TableStyle' object.

Method `createInlineStyle()`**:** Create an inline style that can be used to override a base style. For general use cases, the 'setStyling()' method provides a simpler and more direct way of styling specific parts of a table.

Usage:

```
BasicTable$createInlineStyle(baseStyleName = NULL, declarations = NULL)
```

Arguments:

`baseStyleName` The name of an existing style to base the new style on.

`declarations` CSS style declarations in the form of a list, e.g. 'list("font-weight"="bold", "color"="#0000FF")'

Details: Inline styles are typically used to override the style of some specific cells in a table. Inline styles have no name. In HTML, they are rendered as 'style' attributes on specific table cells, where as named styles are linked to cells using the 'class' attribute.

Returns: The newly created 'TableStyle' object.

Method `setStyling()`**:** Apply styling to a set of cells in the table.

Usage:

```
BasicTable$setStyling(
  rFrom = NULL,
  cFrom = NULL,
  rTo = NULL,
  cTo = NULL,
  rowNumbers = NULL,
  columnNumbers = NULL,
  cells = NULL,
  cellType = NULL,
  visible = NULL,
  baseStyleName = NULL,
  style = NULL,
  declarations = NULL,
  applyBorderToAdjacentCells = FALSE
)
```

Arguments:

`rFrom` An integer row number that specifies the start row for the styling changes.

`cFrom` An integer column number that specifies the start column for the styling changes.

`rTo` An integer row number that specifies the end row for the styling changes.

`cTo` An integer column number that specifies the end column for the styling changes.

`rowNumbers` An integer vector that specifies the row numbers for the styling changes.

`columnNumbers` An integer vector that specifies the column numbers for the styling changes.

`cells` A list containing 'TableCell' objects.

`cellType` One of the following values that specifies the type of cell: root, rowHeader, columnHeader, cell, total. The cellType controls the default styling that is applied to the cell.

`visible` The cell visibility to apply (‘TRUE‘ or ‘FALSE‘).

`baseStyleName` The name of a style to apply.

`style` A ‘TableStyle‘ object to apply.

`declarations` CSS style declarations to apply in the form of a list, e.g. ‘list("font-weight"="bold", "color"="#0000FF")‘

`applyBorderToAdjacentCells` TRUE to override the border in neighbouring cells, e.g. the left border of the current cell becomes the right border of the cell to the left.

Details: There are five ways to specify the part(s) of a table to apply styling to:
(1) By specifying a list of data groups using the ‘groups‘ argument.
(2) By specifying a list of cells using the ‘cells‘ argument.
(3) By specifying a single cell using the ‘rFrom‘ and ‘cFrom‘ arguments.
(4) By specifying a rectangular cell range using the ‘rFrom‘, ‘cFrom‘, ‘rTo‘ and ‘cTo‘ arguments.
(5) By specifying a vector of rowNumbers and/or columnNumbers. If both rowNumbers and columnNumbers are specified, then the cells at the intersection of the specified row numbers and column numbers are styled.
If both rFrom/rTo and rowNumbers are specified, then rFrom/rTo constrain the row numbers specified in rowNumbers.
If both cFrom/cTo and columnNumbers are specified, then cFrom/cTo constrain the column numbers specified in columnNumbers.
See the "Styling" and "Finding and Formatting" vignettes for more information and many examples.

Returns: No return value.

Method `mapStyling()`**:** Apply styling to table cells based on the value of each cell.

Usage:

```
BasicTable$mapStyling(
  styleProperty = NULL,
  cells = NULL,
  valueType = "text",
  mapType = "range",
  mappings = NULL,
  styleLowerValues = FALSE,
  styleHigherValues = TRUE
)
```

Arguments:

`styleProperty` The name of the style property to set on the specified cells, e.g. backgroundcolor.

`cells` A list containing ‘TableCell‘ objects.

`valueType` The type of style value to be set. Must be one of: "text", "character", "number", "numeric", "color" or "colour".
"text" and "character" are equivalent. "number" and "numeric" are equivalent. "color" and "colour" are equivalent.

`mapType` The type of mapping to be performed. The following mapping types are supported:
(1) "value" = a 1:1 mapping which maps each specified "from" value to the corresponding "to" value, e.g. 100 -> "green".
(2) "logic" = each from value is logical criteria. See details.
(3) "range" = values between each pair of "from" values are mapped to the corresponding "to" value, e.g. values in the range 80-100 -> "green" (more specifically values greater than or equal to 80 and less than 100).
(4) "continuous" = rescales values between each pair of "from" values into the range of the corresponding pair of "to" values, e.g. if the "from" range is 80-100 and the corresponding "to" range is 0.8-1, then 90 -> 0.9.
"continuous" cannot be used with valueType="text"/"character".

`mappings` The mappings to be applied, specified in one of the following three forms:
(1) a list containing pairs of values, e.g. 'list(0, "red", 0.4, "yellow", 0.8, "green")'.
(2) a list containing "from" and "to" vectors/lists, e.g. 'list(from=c(0, 0.4, 0.8), to=c("red", "yellow", "green"))'.
(3) a custom mapping function that will be invoked once per cell, e.g. 'function(v, cell) { if(isTRUE(v>0.8)) return("green") }'.
Mappings must be specified in ascending order when valueType="range" or valueType="continuous".
If a custom mapping function is specified, then the valueType and mapType parameters are ignored.

`styleLowerValues` A logical value, default 'FALSE', that specifies whether values less than the lowest specified "from" value should be styled using the style specified for the lowest "from" value. Only applies when valueType="range" or valueType="continuous".

`styleHigherValues` A logical value, default 'TRUE', that specifies whether values greater than the highest specified "from" value should be styled using the style specified for the highest "from" value. Only applies when valueType="range" or valueType="continuous".

Details: 'mapStyling()' is typically used to conditionally apply styling to cells based on the value of each individual cell, e.g. cells with values less than a specified number could be coloured red.
mapType="logic" maps values matching specified logical criteria to specific "to" values. The logical criteria can be any of the following forms (the first matching mapping is used):
(1) a specific value, e.g. 12.
(2) a specific value equality condition, e.g. "v==12", where v represents the cell value.
(3) a value range expression using the following abbreviated form: "value1<=v<value2", e.g. "10<=v<15". Only "<" or "<=" can be used in these value range expressions.
(4) a standard R logical expression, e.g. "10<=v && v<15".
Basic R functions that test the value can also be used, e.g. is.na(v).
See the "Styling" and Finding and Formatting" vignettes for more information and many examples.

Returns: No return value.

Method `resetCells()`**:** Clear the cells of the table.

Usage:

```
BasicTable$resetCells()
```

Details: The cells are reset automatically when structural changes are made to the table, so this method often doesn't needs to be called explicitly.

Returns: No return value.

Method `getCells()`**:** Retrieve cells by a combination of row and/or column numbers. See the "Finding and Formatting" vignette for graphical examples.

Usage:

```
BasicTable$getCells(
  specifyCellsAsList = TRUE,
  rowNumbers = NULL,
  columnNumbers = NULL,
  cellCoordinates = NULL,
  excludeEmptyCells = FALSE,
  matchMode = "simple"
)
```

Arguments:

`specifyCellsAsList` Specify how cells are retrieved. Default 'TRUE'. More information is provided in the details section.

`rowNumbers` A vector of row numbers that specify the rows or cells to retrieve.

`columnNumbers` A vector of column numbers that specify the columns or cells to retrieve.

`cellCoordinates` A list of two-element vectors that specify the coordinates of cells to retrieve. Ignored when 'specifyCellsAsList=FALSE'.

`excludeEmptyCells` Default 'FALSE'. Specify 'TRUE' to exclude empty cells.

`matchMode` Either "simple" (default) or "combinations":

"simple" specifies that row and column arguments are considered separately (logical OR), e.g. rowNumbers=1 and columnNumbers=2 will match all cells in row 1 and all cells in column 2.

"combinations" specifies that row and column arguments are considered together (logical AND), e.g. rowNumbers=1 and columnNumbers=2 will match only the cell single at location (1, 2).

Arguments 'rowNumbers' and 'columnNumbers' are affected by the match mode. All other arguments are not.

Details: When 'specifyCellsAsList=TRUE' (the default):
Get one or more rows by specifying the row numbers as a vector as the rowNumbers argument and leaving the columnNumbers argument set to the default value of 'NULL', or
Get one or more columns by specifying the column numbers as a vector as the columnNumbers argument and leaving the rowNumbers argument set to the default value of 'NULL', or
Get one or more individual cells by specifying the cellCoordinates argument as a list of vectors of length 2, where each element in the list is the row and column number of one cell,
e.g. 'list(c(1, 2), c(3, 4))' specifies two cells, the first located at row 1, column 2 and the second located at row 3, column 4.
When 'specifyCellsAsList=FALSE':
Get one or more rows by specifying the row numbers as a vector as the rowNumbers argument and leaving the columnNumbers argument set to the default value of 'NULL', or
Get one or more columns by specifying the column numbers as a vector as the columnNumbers argument and leaving the rowNumbers argument set to the default value of 'NULL', or
Get one or more cells by specifying the row and column numbers as vectors for the rowNumbers and columnNumbers arguments, or
a mixture of the above, where for entire rows/columns the element in the other vector is set to 'NA', e.g. to retrieve whole rows, specify the row numbers as the rowNumbers but set the corresponding elements in the columnNumbers vector to 'NA'.

Returns: A list of ‘TableCell‘ objects.

Method `findCells()`**:** Find cells matching specified criteria. See the "Finding and Formatting" vignette for graphical examples.

Usage:

```
BasicTable$findCells(
  minValue = NULL,
  maxValue = NULL,
  exactValues = NULL,
  valueRanges = NULL,
  includeNull = TRUE,
  includeNA = TRUE,
  emptyCells = "include",
  rowNumbers = NULL,
  columnNumbers = NULL,
  cellCoordinates = NULL,
  cells = NULL,
  rowColumnMatchMode = "simple"
)
```

Arguments:

`minValue` A numerical value specifying a minimum value threshold.

`maxValue` A numerical value specifying a maximum value threshold.

`exactValues` A vector or list specifying a set of allowed values.

`valueRanges` A vector specifying one or more value range expressions which the cell values must match. If multiple value range expressions are specified, then the cell value must match any of one the specified expressions. See details.

`includeNull` specify TRUE to include ‘NULL‘ in the matched cells, FALSE to exclude ‘NULL‘ values.

`includeNA` specify TRUE to include ‘NA‘ in the matched cells, FALSE to exclude ‘NA‘ values.

`emptyCells` A word that specifies how empty cells are matched - must be one of "include" (default), "exclude" or "only".

`rowNumbers` A vector of row numbers that specify the rows or cells to constrain the search.

`columnNumbers` A vector of column numbers that specify the columns or cells to constrain the search.

`cellCoordinates` A list of two-element vectors that specify the coordinates of cells to constrain the search.

`cells` A ‘TableCell‘ object or a list of ‘TableCell‘ objects to constrain the scope of the search.

`rowColumnMatchMode` Either "simple" (default) or "combinations":
"simple" specifies that row and column arguments are considered separately (logical OR), e.g. rowNumbers=1 and columnNumbers=2 will match all cells in row 1 and all cells in column 2.
"combinations" specifies that row and column arguments are considered together (logical AND), e.g. rowNumbers=1 and columnNumbers=2 will match only the cell single at location (1, 2).
Arguments ‘rowNumbers‘, ‘columnNumbers‘, ‘rowGroups‘ and ‘columnGroups‘ are affected by the match mode. All other arguments are not.

Details: The valueRanges parameter can be any of the following forms:
(1) a specific value, e.g. 12.
(2) a specific value equality condition, e.g. "v==12", where v represents the cell value.
(3) a value range expression using the following abbreviated form: "value1<=v<value2", e.g. "10<=v<15". Only "<" or "<=" can be used in these value range expressions.
(4) a standard R logical expression, e.g. "10<=v && v<15".
Basic R functions that test the value can also be used, e.g. is.na(v).

Returns: A list of 'TableCell' objects.

Method `print()`**:** Outputs a plain text representation of the table to the console or returns a character representation of the table.

Usage:

```
BasicTable$print(asCharacter = FALSE)
```

Arguments:

`asCharacter` 'FALSE' (default) outputs to the console, specify 'TRUE' to instead return a character value (does not output to console).

Returns: Plain text representation of the table.

Method `asMatrix()`**:** Convert the table to a matrix, with or without headings.

Usage:

```
BasicTable$asMatrix(
  firstRowAsColumnNames = FALSE,
  firstColumnAsRowNames = FALSE,
  rawValue = FALSE
)
```

Arguments:

`firstRowAsColumnNames` 'TRUE' to use the first row of the table as the column names in the matrix. Default value 'FALSE'.

`firstColumnAsRowNames` 'TRUE' to use the first column of the table as the row names in the matrix. Default value 'FALSE'.

`rawValue` 'FALSE' (default) outputs the formatted (character) values. Specify 'TRUE' to output the raw cell values.

Details: See the "Outputs" vignette for a comparison of outputs.

Returns: A matrix.

Method `asDataFrame()`**:** Convert the table to a data frame, with or without headings.

Usage:

```
BasicTable$asDataFrame(
  firstRowAsColumnNames = FALSE,
  firstColumnAsRowNames = FALSE,
  rawValue = FALSE,
  stringsAsFactors = NULL
)
```

Arguments:

`firstRowAsColumnNames` 'TRUE' to use the first row of the table as the column names in the data frame Default value 'FALSE'.

`firstColumnAsRowNames` 'TRUE' to use the first column of the table as the row names in the data frame. Default value 'FALSE'.

`rawValue` 'FALSE' (default) outputs the formatted (character) values. Specify 'TRUE' to output the raw cell values.

`stringsAsFactors` Specify 'TRUE' to convert strings to factors, default is 'default.stringsAsFactors()' for R < 4.1.0 and 'FALSE' for R >= 4.1.0.

Details: See the "Outputs" vignette for a comparison of outputs.

Returns: A matrix.

Method `getCss()`**:** Get the CSS declarations for the table.

Usage:

```
BasicTable$getCss(styleNamePrefix = NULL)
```

Arguments:

`styleNamePrefix` A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple tables exist.

Details: See the "Outputs" vignette for more details and examples.

Returns: A character value containing the CSS style declaration.

Method `getHtml()`**:** Generate a HTML representation of the table.

Usage:

```
BasicTable$getHtml(styleNamePrefix = NULL)
```

Arguments:

`styleNamePrefix` A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple tables exist.

Details: See the "Outputs" vignette for more details and examples.

Returns: A list containing HTML tags from the 'htmltools' package. Convert this to a character variable using 'as.character()'.

Method `saveHtml()`**:** Save a HTML representation of the table to file.

Usage:

```
BasicTable$saveHtml(
  filePath = NULL,
  fullPageHTML = TRUE,
  styleNamePrefix = NULL
)
```

Arguments:

`filePath` The file to save the HTML to.

`fullPageHTML` 'TRUE' (default) includes basic HTML around the table HTML so that the result file is a valid HTML file.

`styleNamePrefix` A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple tables exist.

Details: See the "Outputs" vignette for more details and examples.

Returns: No return value.

Method `renderTable()`**:** Render a HTML representation of the table as an HTML widget.

Usage:

```
BasicTable$renderTable(width = NULL, height = NULL, styleNamePrefix = NULL)
```

Arguments:

`width` The width of the widget.

`height` The height of the widget.

`styleNamePrefix` A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple tables exist.

Details: See the "Outputs" vignette for more details and examples.

Returns: A HTML widget from the ‘htmlwidgets‘ package.

Method `writeToExcelWorksheet()`**:** Write the table into the specified workbook and worksheet at the specified row-column location.

Usage:

```
BasicTable$writeToExcelWorksheet(
  wb = NULL,
  wsName = NULL,
  topRowNumber = NULL,
  leftMostColumnNumber = NULL,
  outputValuesAs = "rawValue",
  useFormattedValueIfRawValueIsNull = TRUE,
  applyStyles = TRUE,
  mapStylesFromCSS = TRUE
)
```

Arguments:

`wb` A ‘Workbook‘ object representing the Excel file being written to.

`wsName` A character value specifying the name of the worksheet to write to.

`topRowNumber` An integer value specifying the row number in the Excel worksheet to write the table.

`leftMostColumnNumber` An integer value specifying the column number in the Excel worksheet to write the table.

`outputValuesAs` Must be one of "rawValue" (default), "formattedValueAsText" or "formattedValueAsNumber" to specify how cell values are written into the Excel sheet.

`useFormattedValueIfRawValueIsNull` ‘TRUE‘ to use the formatted cell value instead of the raw cell value if the raw value is ‘NULL‘. ‘FALSE‘ to always use the raw value. Default ‘TRUE‘.

`applyStyles` Default ‘TRUE‘ to write styling information to cells.

`mapStylesFromCSS` Default ‘TRUE‘ to automatically convert CSS style declarations to their Excel equivalents.

Details: See the Excel Output vignette for more details.

Returns: No return value.

Method `asFlexTable()`**:** Convert table to a flextable table..

Usage:

```
BasicTable$asFlexTable(applyStyles = TRUE, mapStylesFromCSS = TRUE)
```

Arguments:

`applyStyles` Default 'TRUE' to write styling information for cells.

`mapStylesFromCSS` Default 'TRUE' to automatically convert CSS style declarations to their flextable equivalents.

Details: See the Outputs vignette for more details.

Returns: A table from the flextable package.

Method `trace()`**:** Capture a call for tracing purposes. This is an internal method.

Usage:

```
BasicTable$trace(methodName, desc, detailList = NULL)
```

Arguments:

`methodName` The name of the method being invoked.

`desc` Short description of method call.

`detailList` A list containing detail such as parameter values.

Returns: No return value.

Method `asList()`**:** Return the contents of the table as a list for debugging.

Usage:

```
BasicTable$asList()
```

Returns: A list of various object properties..

Method `asJSON()`**:** Return the contents of the table as JSON for debugging.

Usage:

```
BasicTable$asJSON()
```

Returns: A JSON representation of various object properties.

Method `viewJSON()`**:** Use the 'listviewer' package to view the table as JSON for debugging.

Usage:

```
BasicTable$viewJSON()
```

Returns: No return value.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
BasicTable$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# The package vignettes have many more examples of working with the
# BasicTable class.
# Quickly rendering a table as an htmlwidget:
library(basictabler)
qhtbl(data.frame(a=1:2, b=3:4))
# Rendering a larger table as an htmlwidget:
library(basictabler)
library(dplyr)
tocsummary <- bhmsummary %>%
  group_by(TOC) %>%
  summarise(OnTimeArrivals=sum(OnTimeArrivals),
            OnTimeDepartures=sum(OnTimeDepartures),
            TotalTrains=sum(TrainCount)) %>%
  ungroup() %>%
  mutate(OnTimeArrivalPercent=OnTimeArrivals/TotalTrains*100,
         OnTimeDeparturePercent=OnTimeDepartures/TotalTrains*100) %>%
  arrange(TOC)

tbl <- BasicTable$new()
columnHeaders <- c("TOC", "On-Time Arrivals", "On-Time Departures",
  "Total Trains", "On-Time Arrival %", "On-Time Departure %")
columnFormats=list()
columnFormats[[2]] <- list(big.mark=",")
columnFormats[[3]] <- list(big.mark=",")
columnFormats[[4]] <- list(big.mark=",")
columnFormats[[5]] <- "%.1f"
columnFormats[[6]] <- "%.1f"
tbl$addData(tocsummary, columnNamesAsColumnHeaders=FALSE,
  firstColumnAsRowHeaders=TRUE,
  explicitColumnHeaders=columnHeaders, columnFormats=columnFormats)
tbl$renderTable()
```

`basictabler` *Render a table as a HTML widget.*

Description

The `basictabler` function is primarily intended for use with Shiny web applications.

Usage

```
basictabler(bt, width = NULL, height = NULL, styleNamePrefix = NULL)
```

Arguments

`bt`	The table to render.
`width`	The target width.
`height`	The target height.

styleNamePrefix
: A text prefix to be prepennded to the CSS declarations (to ensure uniqueness).

Value

A HTML widget.

Examples

```
# See the Shiny vignette in this package for an example.
```

---

basictablerOutput *Standard function for Shiny scaffolding.*

Description

Standard function for Shiny scaffolding.

Usage

```
basictablerOutput(outputId, width = "100%", height = "100%")
```

Arguments

outputId	The id of the html element that will contain the htmlwidget.
width	The target width of the htmlwidget.
height	The target height of the htmlwidget.

Examples

```
# See the Shiny vignette in this package for an example.
```

---

basictablerSample *Generate a sample basic table.*

Description

The `basictablerSample` function generates a sample basic table.

Usage

```
basictablerSample()
```

Value

Example basic table describing the performance of trains run by different train operating companies in Birmingham (UK).

Examples

```
# Generate sample table.
basictablerSample()
```

`bhmsummary` *A Summary of Birmingham Trains, Dec 2016-Feb 2017.*

Description

A dataset summarising all of the trains that either originated at, passed through or terminated at Birmingham New Street railway station in the UK between 1st December 2016 and 28th February 2017 inclusive.

Usage

```
bhmsummary
```

Format

A data frame with 4839 rows and 14 variables:

Status Train status: A=Active, C=Cancelled, R=Reinstated

TOC Train operating company

TrainCategory Express Passenger or Ordinary Passenger

PowerType DMU=Diesel Multiple Unit, EMU=Electrical Multiple Unit, HST=High Speed Train

SchedSpeedMPH Scheduled maximum speed (in miles per hour)

GbttMonth The month of the train in the Great Britain Train Timetable (GBTT)

GbttWeekDate The week of the train in the Great Britain Train Timetable (GBTT)

Origin 3-letter code denoting the scheduled origin of the train

Destination 3-letter code denoting the scheduled destination of the train

TrainCount The number of scheduled trains

OnTimeArrivals The number of trains that arrived in Birmingham on time

OnTimeDepartures The number of trains that departed Birmingham on time

TotalArrivalDelayMinutes The total number of delay minutes for arrivals

TotalDepartureDelayMinutes The total number of delay minutes for departures

Source

https://www.recenttraintimes.co.uk/

checkArgument *Perform basic checks on a function argument.*

Description

checkArgument is a utility function that provides basic assurances about function argument values and generates standardised error messages when invalid values are encountered. This function should not be used outside the basictabler package.

Usage

```
checkArgument(
  argumentCheckMode,
  checkDataTypes,
  className,
  methodName,
  argumentValue,
  isMissing,
  allowMissing = FALSE,
  allowNull = FALSE,
  allowedClasses = NULL,
  mustBeAtomic = FALSE,
  allowedListElementClasses = NULL,
  listElementsMustBeAtomic = FALSE,
  allowedValues = NULL,
  minValue = NULL,
  maxValue = NULL,
  maxLength = NULL
)
```

Arguments

argumentCheckMode	A number between 0 and 4 specifying the checks to perform.
checkDataTypes	A logical value specifying whether the data types should be checked when argumentCheckMode=3
className	The name of the calling class, for inclusion in error messages.
methodName	The name of the calling method, for inclusion in error messages.
argumentValue	The value to check.
isMissing	Whether the argument is missing in the calling function.
allowMissing	Whether missing values are permitted.
allowNull	Whether null values are permitted.
allowedClasses	The names of the allowed classes for argumentValue.
mustBeAtomic	Whether the argument value must be atomic.

`allowedListElementClasses`
: For argument values that are lists(), the names of the allowed classes for the elements in the list.

`listElementsMustBeAtomic`
: For argument values that are lists(), whether the list elements must be atomic.

`allowedValues` For argument values that must be one value from a set list, the list of allowed values.

`minValue` For numerical values, the lowest allowed value.

`maxValue` For numerical values, the highest allowed value.

`maxLength` For character values, the maximum allowed length (in characters) of the value.

Value

No return value. If invalid values are encountered, the `stop()` function is used to interrupt execution.

`cleanCssValue` *Cleans up a CSS attribute value.*

Description

`cleanCssValue` is a utility function that performs some basic cleanup on CSS attribute values. Leading and trailing whitespace is removed. The CSS values "initial" and "inherit" are blocked. The function is vectorised so can be used with arrays.

Usage

```
cleanCssValue(cssValue)
```

Arguments

`cssValue` The value to cleanup.

Value

The cleaned value.

containsText *Check whether a text value is present in another text value.*

Description

containsText is a utility function returns TRUE if one text value is present in another. Case sensitive. If textToSearch is a vector, returns TRUE if any element contains textToFind.

Usage

```
containsText(textToSearch, textToFind)
```

Arguments

textToSearch	The value to be searched.
textToFind	The value to find.

Value

TRUE if the textToFind value is found.

FlexTableRenderer *R6 class that converts a table to a flextable from the 'flextabler' package.*

Description

The 'FlexTableRenderer' class creates a representation of a table using the 'flextable' package. See the Output vignette for more details.

Format

R6Class object.

Methods

Public methods:

- FlexTableRenderer$new()
- FlexTableRenderer$writeToCell()
- FlexTableRenderer$writeBorder()
- FlexTableRenderer$asFlexTable()
- FlexTableRenderer$clone()

Method new(): Create a new 'FlexTableRenderer' object.

Usage:

```
FlexTableRenderer$new(parentTable)
```

Arguments:

`parentTable` Owning table.

Returns: No return value.

Method `writeToCell()`**:** Write a value to a cell, optionally with styling and cell merging.

Usage:

```
FlexTableRenderer$writeToCell(
  ft = NULL,
  rowNumber = NULL,
  columnNumber = NULL,
  totalRowCount = NULL,
  totalColumnCount = NULL,
  value = NULL,
  applyStyles = TRUE,
  baseStyleName = NULL,
  style = NULL,
  mapFromCss = TRUE,
  mergeRows = NULL,
  mergeColumns = NULL
)
```

Arguments:

`ft` The flextable to write to.

`rowNumber` The row number of the cell where the value is to be written.

`columnNumber` The column number of the cell where the value is to be written.

`totalRowCount` Used when writing the cell border.

`totalColumnCount` Used when writing the cell border.

`value` The value to be written. Since the flextable is created from a data frame, this argument can be omitted.

`applyStyles` 'TRUE' (default) to also set the styling of the cell, 'FALSE' to only write the value.

`baseStyleName` The name of the style from the table theme to apply to the cell.

`style` A 'TableStyle' object that contains additional styling to apply to the cell.

`mapFromCss` 'TRUE' (default) to map the basictabler CSS styles to corresponding Excel styles, 'FALSE' to apply only the specified xl styles.

`mergeRows` If the cell is to be merged with adjacent cells, then an integer or numeric vector specifying the row numbers of the merged cell. NULL (default) to not merge cells.

`mergeColumns` If the cell is to be merged with adjacent cells, then an integer or numeric vector specifying the column numbers of the merged cell. NULL (default) to not merge cells.

Returns: The updated flextable definition.

Method `writeBorder()`**:** Write the borders of a cell.

Usage:

```
FlexTableRenderer$writeBorder(
  ft = NULL,
  rowNumber = NULL,
  columnNumber = NULL,
  totalRowCount = NULL,
  totalColumnCount = NULL,
  border = NULL,
  borderLeft = NULL,
  borderRight = NULL,
  borderTop = NULL,
  borderBottom = NULL
)
```

Arguments:

`ft` The flextable to write to.

`rowNumber` The row number of the cell where the border is to be written.

`columnNumber` The column number of the cell where the border is to be written.

`totalRowCount` The total number of rows in the table.

`totalColumnCount` The total number of columns in the table.

`border` The border to be applied to all sides of the cell (unless a border is specified using the other arguments).

`borderLeft` The border to apply to the left side of the cell.

`borderRight` The border to apply to the right side of the cell.

`borderTop` The border to apply to the top side of the cell.

`borderBottom` The border to apply to the bottom side of the cell.

Returns: The updated flextable definition.

Method `asFlexTable()`**:** Convert table to a flextable table.

Usage:

```
FlexTableRenderer$asFlexTable(applyStyles = TRUE, mapStylesFromCSS = TRUE)
```

Arguments:

`applyStyles` 'TRUE' (default) to also set the styling of the cells, 'FALSE' to only write the value.

`mapStylesFromCSS` 'TRUE' (default) to map the basictabler CSS styles to corresponding flextable styles where possible, 'FALSE' to apply only the specified ft styles.

Returns: No return value.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FlexTableRenderer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should not be used by end users.  It is an internal class
# created only by the BasicTable class.  It is used when converting to a
# flextable.
```

`FlexTableStyle` *R6 class that specifies styling as used by the 'flextable' package.*

Description

The 'FlexTableStyle' class specifies the styling for cells in a table from the flextable package.

Format

`R6Class` object.

Active bindings

`baseStyleName` The name of the base style in the table.

`isBaseStyle` 'TRUE' when this style is the equivalent of a named style in the table, 'FALSE' if this style has additional settings over and above the base style of the same name.

`fontName` The name of the font (single font name, not a CSS style list).

`fontSize` The size of the font (units: point).

`bold` TRUE' if text is bold.

`italic` 'TRUE' if text is italic.

`bgColor` The background colour for the cell (as a hex value, e.g. #00FF00).

`textColor` The color of the text (as a hex value).

`hAlign` The horizontal alignment of the text: left, center or right.

`vAlign` The vertical alignment of the text: top, middle or bottom.

`textRotation` The rotation angle of the text or 255 for vertical.

`paddingAll` The padding to apply to all sides of each cell.

`paddingLeft` The padding to apply to the left side of each cell.

`paddingRight` The padding to apply to the right side of each cell.

`paddingTop` The padding to apply to the top of each cell.

`paddingBottom` The padding to apply to the bottom of each cell.

`borderAll` A list (with elements style, color and width) specifying the border settings for all four sides of each cell at once.

`borderLeft` A list (with elements style, color and width) specifying the border settings for the left border of each cell.

`borderRight` A list (with elements style, color and width) specifying the border settings for the right border of each cell.

`borderTop` A list (with elements style, color and width) specifying the border settings for the top border of each cell.

`borderBottom` A list (with elements style, color and width) specifying the border settings for the bottom border of each cell.

Methods

Public methods:

- FlexTableStyle$new()
- FlexTableStyle$isBasicStyleNameMatch()
- FlexTableStyle$isFullStyleDetailMatch()
- FlexTableStyle$asList()
- FlexTableStyle$asJSON()
- FlexTableStyle$asString()
- FlexTableStyle$clone()

Method `new()`**:** Create a new 'FlexTableStyle' object.

Usage:

```
FlexTableStyle$new(
  parentTable,
  baseStyleName = NULL,
  isBaseStyle = NULL,
  fontName = NULL,
  fontSize = NULL,
  bold = NULL,
  italic = NULL,
  bgColor = NULL,
  textColor = NULL,
  hAlign = NULL,
  vAlign = NULL,
  textRotation = NULL,
  paddingAll = NULL,
  paddingLeft = NULL,
  paddingRight = NULL,
  paddingTop = NULL,
  paddingBottom = NULL,
  borderAll = NULL,
  borderLeft = NULL,
  borderRight = NULL,
  borderTop = NULL,
  borderBottom = NULL
)
```

Arguments:

`parentTable` Owning table.

`baseStyleName` The name of the base style in the table.

`isBaseStyle` 'TRUE' when this style is the equivalent of a named style in the table, 'FALSE' if this style has additional settings over and above the base style of the same name.

`fontName` The name of the font (single font name, not a CSS style list).

`fontSize` The size of the font (units: point).

`bold` 'TRUE' if text is bold.

`italic` 'TRUE' if text is italic.

`bgColor` The background colour for the cell (as a hex value, e.g. #00FF00).

`textColor` The color of the text (as a hex value).

`hAlign` The horizontal alignment of the text: left, center or right.

`vAlign` The vertical alignment of the text: top, middle or bottom.

`textRotation` The rotation angle of the text or 255 for vertical.

`paddingAll` The padding to apply to all sides of each cell.

`paddingLeft` The padding to apply to the left side of each cell.

`paddingRight` The padding to apply to the right side of each cell.

`paddingTop` The padding to apply to the top of each cell.

`paddingBottom` The padding to apply to the bottom of each cell.

`borderAll` A list (with elements style, color and width) specifying the border settings for all four sides of each cell at once.

`borderLeft` A list (with elements style, color and width) specifying the border settings for the left border of each cell.

`borderRight` A list (with elements style, color and width) specifying the border settings for the right border of each cell.

`borderTop` A list (with elements style, color and width) specifying the border settings for the top border of each cell.

`borderBottom` A list (with elements style, color and width) specifying the border settings for the bottom border of each cell.

Returns: No return value.

Method `isBasicStyleNameMatch()`**:** Check if this style matches the specified base style name.

Usage:

```
FlexTableStyle$isBasicStyleNameMatch(baseStyleName = NULL)
```

Arguments:

`baseStyleName` The style name to compare to.

Returns: 'TRUE' if the style name matches, 'FALSE' otherwise.

Method `isFullStyleDetailMatch()`**:** Check if this style matches the specified style properties.

Usage:

```
FlexTableStyle$isFullStyleDetailMatch(
  baseStyleName = NULL,
  isBaseStyle = NULL,
  fontName = NULL,
  fontSize = NULL,
  bold = NULL,
  italic = NULL,
  bgColor = NULL,
  textColor = NULL,
  hAlign = NULL,
  vAlign = NULL,
  textRotation = NULL,
  paddingAll = NULL,
```

```
  paddingLeft = NULL,
  paddingRight = NULL,
  paddingTop = NULL,
  paddingBottom = NULL,
  borderAll = NULL,
  borderLeft = NULL,
  borderRight = NULL,
  borderTop = NULL,
  borderBottom = NULL
)
```

Arguments:

`baseStyleName` The style name to compare to.
`isBaseStyle` Whether the style being compared to is a base style.
`fontName` The font name to compare to.
`fontSize` The font size to compare to.
`bold` The style property bold to compare to.
`italic` The style property italic to compare to.
`bgColor` The style property bgColor to compare to.
`textColor` The style property textColor to compare to.
`hAlign` The style property hAlign to compare to.
`vAlign` The style property vAlign to compare to.
`textRotation` The style property textRotation to compare to.
`paddingAll` The padding to apply to all sides of each cell.
`paddingLeft` The padding to apply to the left side of each cell.
`paddingRight` The padding to apply to the right side of each cell.
`paddingTop` The padding to apply to the top of each cell.
`paddingBottom` The padding to apply to the bottom of each cell.
`borderAll` The style property borderAll to compare to.
`borderLeft` The style property borderLeft to compare to.
`borderRight` The style property borderRight to compare to.
`borderTop` The style property borderTop to compare to.
`borderBottom` The style property borderBottom to compare to.
`valueFormat` The style value format to compare to.
`minColumnWidth` The style property minColumnWidth to compare to.
`minRowHeight` The style property minRowHeight to compare to.

Returns: 'TRUE' if the style matches, 'FALSE' otherwise.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
FlexTableStyle$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
FlexTableStyle$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return the contents of this object as a string for debugging.

Usage:

```
FlexTableStyle$asString()
```

Returns: A character representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FlexTableStyle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by using the functions in the table.
# It is not intended to be created by users outside of the table.
```

`FlexTableStyles` *R6 class that defines a collection of styles as used by the 'flextable' package.*

Description

The 'FlexTableStyles' class stores a collection of 'FlexTableStyle' style objects.

Format

`R6Class` object.

Active bindings

`count` The number of styles in the collection.

`styles` A list of 'FlexTableStyle' objects that comprise the collection.

Methods

Public methods:

- `FlexTableStyles$new()`
- `FlexTableStyles$clearStyles()`
- `FlexTableStyles$findNamedStyle()`
- `FlexTableStyles$findOrAddStyle()`
- `FlexTableStyles$addNamedStyles()`

- FlexTableStyles$asList()
- FlexTableStyles$asJSON()
- FlexTableStyles$asString()
- FlexTableStyles$clone()

Method `new()`**:** Create a new 'FlexTableStyles' object.

Usage:

```
FlexTableStyles$new(parentTable)
```

Arguments:

`parentTable` Owning table.

Returns: No return value.

Method `clearStyles()`**:** Clear the collection removing all styles.

Usage:

```
FlexTableStyles$clearStyles()
```

Returns: No return value.

Method `findNamedStyle()`**:** Find a style in the collection matching the specified base style name.

Usage:

```
FlexTableStyles$findNamedStyle(baseStyleName)
```

Arguments:

`baseStyleName` The style name to find.

Returns: A 'TableTableFlexTbl' object that is the style matching the specified base style name or 'NULL' otherwise.

Method `findOrAddStyle()`**:** Find a style in the collection matching the specified base style name and style properties. If there is no matching style, then optionally add a new style.

Usage:

```
FlexTableStyles$findOrAddStyle(
  action = "findOrAdd",
  baseStyleName = NULL,
  isBaseStyle = NULL,
  style = NULL,
  mapFromCss = TRUE
)
```

Arguments:

`action` The action to carry out. Must be one of "find", "add" or "findOrAdd" (default).

`baseStyleName` The style name to find/add.

`isBaseStyle` Is the style to be found/added a base style?

`style` A 'TableStyle' object specifying style properties to be found/added.

`mapFromCss` 'TRUE' (default) to map the basictabler CSS styles to corresponding flextable styles, 'FALSE' to apply only the specified ft styles.

Returns: A 'TableTableFlexTbl' object that is the style matching the specified base style name or 'NULL' otherwise.

Method `addNamedStyles()`**:** Populate the FlexTbl styles based on the styles defined in the table.

Usage:

```
FlexTableStyles$addNamedStyles(mapFromCss = TRUE)
```

Arguments:

`mapFromCss` 'TRUE' (default) to map the basictabler CSS styles to corresponding Excel styles, 'FALSE' to apply only the specified ft styles.

Returns: No return value.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
FlexTableStyles$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
FlexTableStyles$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return the contents of this object as a string for debugging.

Usage:

```
FlexTableStyles$asString(seperator = ", ")
```

Arguments:

`seperator` Delimiter used to combine multiple values into a string.

Returns: A character representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FlexTableStyles$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should not be used by end users.  It is an internal class
# created only by the BasicTable class.  It is used when converting to a
# flextable.
```

getBlankTblTheme *Get an empty theme for applying no styling to a table.*

Description

Get an empty theme for applying no styling to a table.

Usage

```
getBlankTblTheme(parentTable, themeName = "blank")
```

Arguments

parentTable	Owning table.
themeName	The name to use as the new theme name.

Value

A TableStyles object.

getCompactTblTheme *Get the compact theme for styling a table.*

Description

Get the compact theme for styling a table.

Usage

```
getCompactTblTheme(parentTable, themeName = "compact")
```

Arguments

parentTable	Owning table.
themeName	The name to use as the new theme name.

Value

A TableStyles object.

`getDefaultTblTheme` *Get the default theme for styling a table.*

Description

Get the default theme for styling a table.

Usage

```
getDefaultTblTheme(parentTable, themeName = "default")
```

Arguments

`parentTable`	Owning table.
`themeName`	The name to use as the new theme name.

Value

A TableStyles object.

`getFtBorderFromCssBorder` *Convert CSS border values to those used by the flextable package.*

Description

`getXlBorderFromCssBorder` parses the CSS combined border declarations (i.e. border, border-left, border-right, border-top, border-bottom) and returns a list containing an openxlsx border style and color as separate elements.

Usage

```
getFtBorderFromCssBorder(text)
```

Arguments

`text`	The border declaration to parse.

Value

A list containing two elements: width, style and color.

getFtBorderStyleFromCssBorder

Convert CSS border style values to those used by the flextable package.

Description

`getFtBorderStyleFromCssBorder` takes border parameters expressed as a list (must contain an element named style) and returns a border style that is compatible with the flextable package.

Usage

```
getFtBorderStyleFromCssBorder(border)
```

Arguments

`border`	A list containing an element named style.

Value

A flextable border style.

getFtBorderWidthFromCssBorder

Convert CSS border width to those used by the flextable package.

Description

`getFtBorderStyleFromCssBorder` takes border parameters expressed as a list (must contain an element named style) and returns a border style that is compatible with the flextable package.

Usage

```
getFtBorderWidthFromCssBorder(border)
```

Arguments

`border`	A list containing an element named style.

Value

A flextable border style.

getLargePlainTblTheme *Get the large plain theme for styling a table.*

Description

Get the large plain theme for styling a table.

Usage

```
getLargePlainTblTheme(parentTable, themeName = "largeplain")
```

Arguments

parentTable	Owning table.
themeName	The name to use as the new theme name.

Value

A TableStyles object.

getNextPosition *Find the first value in an array that is larger than the specified value.*

Description

getNextPosition is a utility function that helps when parsing strings that contain delimiters.

Usage

```
getNextPosition(positions, afterPosition)
```

Arguments

positions	An ordered numeric vector.
afterPosition	The value to start searching after.

Value

The first value in the array larger than afterPosition.

`getSimpleColoredTblTheme` *Get a simple coloured theme.*

Description

Get a simple coloured theme that can be used to style a table into a custom colour scheme.

Usage

```
getSimpleColoredTblTheme(
  parentTable,
  themeName = "coloredTheme",
  colors = NULL,
  fontName = NULL,
  theme = NULL
)
```

Arguments

`parentTable`	Owning table.
`themeName`	The name to use as the new theme name.
`colors`	The set of colours to use when generating the theme (see the Styling vignette for details). This parameter exists for backward compatibility.
`fontName`	The name of the font to use, or a comma separated list (for font-fall-back). This parameter exists for backward compatibility.
`theme`	A simple theme specified in the form of a list. See example for supported list elements (all other elements will be ignored).

Value

A TableStyles object.

Examples

```
simpleOrangeTheme <- list(
  fontName="Verdana, Arial",
  fontSize="0.75em",
  headerBackgroundColor = "rgb(237, 125, 49)",
  headerColor = "rgb(255, 255, 255)",
  cellBackgroundColor = "rgb(255, 255, 255)",
  cellColor = "rgb(0, 0, 0)",
  totalBackgroundColor = "rgb(248, 198, 165)",
  totalColor = "rgb(0, 0, 0)",
  borderColor = "rgb(198, 89, 17)"
)
library(basictabler)
```

```
tbl <- qtbl(data.frame(a=1:2, b=3:4))
# then
tbl$theme <- simpleOrangeTheme
# or
theme <- getSimpleColoredTblTheme(tbl, theme=simpleOrangeTheme)
# make further changes to the theme
tbl$theme <- simpleOrangeTheme
# theme set, now render table
tbl$renderTable()
```

`getTblTheme` *Get a built-in theme for styling a table.*

Description

`getTblTheme` returns the specified theme.

Usage

```
getTblTheme(parentTable, themeName = NULL)
```

Arguments

`parentTable`	Owning table.
`themeName`	The name of the theme to retrieve.

Value

A TableStyles object.

Examples

```
library(basictabler)
tbl <- qtbl(data.frame(a=1:2, b=3:4))
tbl$theme <- getTblTheme(tbl, "largeplain")
tbl$renderTable()
```

getXlBorderFromCssBorder *Convert CSS border style values to those used by the openxlsx package.*

Description

getXlBorderFromCssBorder parses the CSS combined border declarations (i.e. border, border-left, border-right, border-top, border-bottom) and returns a list containing an openxlsx border style and color as separate elements.

Usage

```
getXlBorderFromCssBorder(text)
```

Arguments

text	The border declaration to parse.

Value

A list containing two elements: style and color.

getXlBorderStyleFromCssBorder *Convert CSS border values to those used by the openxlsx package.*

Description

getXlBorderStyleFromCssBorder takes border parameters expressed as a list (containing elements: width and style) and returns a border style that is compatible with the openxlsx package.

Usage

```
getXlBorderStyleFromCssBorder(border)
```

Arguments

border	A list containing elements width and style.

Value

An openxlsx border style.

isNumericValue *Check whether a numeric value is present.*

Description

isNumericValue is a utility function returns TRUE only when a numeric value is present. NULL, NA, numeric(0) and integer(0) all return FALSE.

Usage

```
isNumericValue(value)
```

Arguments

value	The value to check.

Value

TRUE if a numeric value is present.

isTextValue *Check whether a text value is present.*

Description

isTextValue is a utility function returns TRUE only when a text value is present. NULL, NA, character(0) and "" all return FALSE.

Usage

```
isTextValue(value)
```

Arguments

value	The value to check.

Value

TRUE if a non-blank text value is present.

oneToNULL *Convert a value of 1 to a NULL value.*

Description

oneToNull is a utility function that returns NULL when a value of 0 or 1 is passed to it, otherwise it returns the original value.

Usage

```
oneToNULL(value, convertOneToNULL)
```

Arguments

value	The value to check.
convertOneToNULL	TRUE to convert 1 to NULL.

Value

NULL if value==1, otherwise value.

parseColor *Convert a CSS colour into a hex based colour code.*

Description

parseColor converts a colour value specified in CSS to a hex based colour code. Example supported input values/formats/named colours are: #0080FF, rgb(0, 128, 255), rgba(0, 128, 255, 0.5) and red, green, etc.

Usage

```
parseColor(color)
```

Arguments

color	The colour to convert.

Value

The colour as a hex code, e.g. #FF00A0.

parseCssBorder *Parse a CSS border value.*

Description

parseCssBorder parses the CSS combined border declarations (i.e. border, border-left, border-right, border-top, border-bottom) and returns a list containing the width, style and color as separate elements.

Usage

```
parseCssBorder(text)
```

Arguments

text	The border declaration to parse.

Value

A list containing three elements: width, style and color.

parseCssSizeToPt *Convert a CSS size value into points.*

Description

parseCssSizeToPt will take a CSS style and convert it to points. Supported input size units are in, cm, mm, pt, pc, px, em, are converted exactly: in, cm, mm, pt, pc: using 1in = 2.54cm = 25.4mm = 72pt = 6pc. The following are converted approximately: px, em, approx 1em=16px=12pt and 100

Usage

```
parseCssSizeToPt(size)
```

Arguments

size	A size specified in common CSS units.

Value

The size converted to points.

parseCssSizeToPx *Convert a CSS size value into pixels*

Description

parseCssSizeToPx will take a CSS style and convert it to pixels Supported input size units are in, cm, mm, pt, pc, px, em, are converted exactly: in, cm, mm, pt, pc: using 1in = 2.54cm = 25.4mm = 72pt = 6pc. The following are converted approximately: px, em, approx 1em=16px=12pt and 100

Usage

```
parseCssSizeToPx(size)
```

Arguments

size	A size specified in common CSS units.

Value

The size converted to pixels.

parseCssString *Split a CSS attribute value into a vector/array.*

Description

parseCssString is a utility function that splits a string into a vector/array. The function pays attention to text qualifiers (single and double quotes) so won't split if the delimiter occurs inside a value.

Usage

```
parseCssString(text, separator = ",", removeEmptyString = TRUE)
```

Arguments

text	The text to split.
separator	The field separator, default comma.
removeEmptyString	TRUE to not return empty string / whitespace values.

Value

An R vector containing the values from text split up.

parseFtBorder *Parse an ft-border value.*

Description

parseFtBorder parses the combined ft border declarations (i.e. ft-border, ft-border-left, ft-border-right, ft-border-top, ft-border-bottom) and returns a list containing width, style and color as separate elements.

Usage

```
parseFtBorder(text)
```

Arguments

text	The border declaration to parse.

Value

A list containing two elements: width, style and color.

parseXlBorder *Parse an xl-border value.*

Description

parseXlBorder parses the combined xl border declarations (i.e. xl-border, xl-border-left, xl-border-right, xl-border-top, xl-border-bottom) and returns a list containing style and color as separate elements.

Usage

```
parseXlBorder(text)
```

Arguments

text	The border declaration to parse.

Value

A list containing two elements: style and color.

PxToPt *Convert a number of pixels to points*

Description

PxToPt converts pixels to points. 1inch = 72pt = 96px according to W3C. Ref: https://www.w3.org/TR/css3-values/#absolute-lengths

Usage

```
PxToPt(px)
```

Arguments

px	The number of pixels convert.

Value

The corresponding number of points.

---

qhtbl *Quickly render a basic table in HTML.*

Description

The qhpvt function renders a basic table as a HTML widget with one line of R.

Usage

```
qhtbl(
  dataFrameOrMatrix,
  columnNamesAsColumnHeaders = TRUE,
  explicitColumnHeaders = NULL,
  rowNamesAsRowHeaders = FALSE,
  firstColumnAsRowHeaders = FALSE,
  explicitRowHeaders = NULL,
  numberOfColumnsAsRowHeaders = 0,
  columnFormats = NULL,
  columnCellTypes = NULL,
  theme = NULL,
  replaceExistingStyles = FALSE,
  tableStyle = NULL,
  headingStyle = NULL,
  cellStyle = NULL,
  totalStyle = NULL,
  ...
)
```

Arguments

`dataFrameOrMatrix`
: The data frame or matrix containing the data to be displayed in the table.

`columnNamesAsColumnHeaders`
: TRUE to use the data frame column names as the column headers in the table.

`explicitColumnHeaders`
: A character vector of column headers.

`rowNamesAsRowHeaders`
: TRUE to use the data frame row names as the row headers in the table.

`firstColumnAsRowHeaders`
: TRUE to use the first column in the data frame as row headings.

`explicitRowHeaders`
: A character vector of row headers.

`numberOfColumnsAsRowHeaders`
: The number of columns to be set as row headers.

`columnFormats`
: A list containing format specifiers, each of which is either an sprintf() character value, a list of format() arguments or an R function that provides custom formatting logic.

`columnCellTypes`
: A vector that is the same length as the number of columns in the data frame, where each element is one of the following values that specifies the type of cell: root, rowHeader, columnHeader, cell, total. The cellType controls the default styling that is applied to the cell. Typically only rowHeader, cell or total would be used.

`theme`
: Either the name of a built-in theme (default, largeplain, compact or blank/none) or a list which specifies the default formatting for the table.

`replaceExistingStyles`
: TRUE to completely replace the default styling with the specified tableStyle, headingStyle, cellStyle and/or totalStyle

`tableStyle`
: A list of CSS style declarations that apply to the table.

`headingStyle`
: A list of CSS style declarations that apply to the heading cells in the table.

`cellStyle`
: A list of CSS style declarations that apply to the normal cells in the table.

`totalStyle`
: A list of CSS style declarations that apply to the total cells in the table.

`...`
: Additional arguments, currently compatibility, argumentCheckMode and/or styleNamePrefix.

Value

A basic table.

Examples

```
qhtbl(bhmsummary[1:5, c("GbttWeekDate", "Origin", "Destination", "TrainCount",
  "OnTimeArrivals")])
qhtbl(bhmsummary[1:5, c("GbttWeekDate", "Origin", "Destination", "TrainCount",
  "OnTimeArrivals")], columnNamesAsColumnHeaders=FALSE,
  explicitColumnHeaders=c("Week", "From", "To", "Trains", "On-Time"))
```

qtbl *Quickly build a basic table.*

Description

The qtbl function builds a basic table with one line of R.

Usage

```
qtbl(
  dataFrameOrMatrix,
  columnNamesAsColumnHeaders = TRUE,
  explicitColumnHeaders = NULL,
  rowNamesAsRowHeaders = FALSE,
  firstColumnAsRowHeaders = FALSE,
  explicitRowHeaders = NULL,
  numberOfColumnsAsRowHeaders = 0,
  columnFormats = NULL,
  columnCellTypes = NULL,
  theme = NULL,
  replaceExistingStyles = FALSE,
  tableStyle = NULL,
  headingStyle = NULL,
  cellStyle = NULL,
  totalStyle = NULL,
  ...
)
```

Arguments

dataFrameOrMatrix
: The data frame or matrix containing the data to be displayed in the table.

columnNamesAsColumnHeaders
: TRUE to use the data frame column names as the column headers in the table.

explicitColumnHeaders
: A character vector of column headers.

rowNamesAsRowHeaders
: TRUE to use the data frame row names as the row headers in the table.

firstColumnAsRowHeaders
: TRUE to use the first column in the data frame as row headings.

explicitRowHeaders
: A character vector of row headers.

numberOfColumnsAsRowHeaders
: The number of columns to be set as row headers.

columnFormats
: A list containing format specifiers, each of which is either an sprintf() character value, a list of format() arguments or an R function that provides custom formatting logic.

columnCellTypes
: A vector that is the same length as the number of columns in the data frame, where each element is one of the following values that specifies the type of cell: root, rowHeader, columnHeader, cell, total. The cellType controls the default styling that is applied to the cell. Typically only rowHeader, cell or total would be used.

theme
: Either the name of a built-in theme (default, largeplain, compact or blank/none) or a list which specifies the default formatting for the table.

replaceExistingStyles
: TRUE to completely replace the default styling with the specified tableStyle, headingStyle, cellStyle and/or totalStyle

tableStyle
: A list of CSS style declarations that apply to the table.

headingStyle
: A list of CSS style declarations that apply to the heading cells in the table.

cellStyle
: A list of CSS style declarations that apply to the normal cells in the table.

totalStyle
: A list of CSS style declarations that apply to the total cells in the table.

...
: Additional arguments, currently compatibility and/or argumentCheckMode.

Value

A basic table.

Examples

```
qtbl(bhmsummary[1:5, c("GbttWeekDate", "Origin", "Destination", "TrainCount",
  "OnTimeArrivals")])
qtbl(bhmsummary[1:5, c("GbttWeekDate", "Origin", "Destination", "TrainCount",
  "OnTimeArrivals")], columnNamesAsColumnHeaders=FALSE,
  explicitColumnHeaders=c("Week", "From", "To", "Trains", "On-Time"))
```

---

renderBasictabler *Standard function for Shiny scaffolding.*

Description

Standard function for Shiny scaffolding.

Usage

```
renderBasictabler(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

expr
: The R expression to execute and render in the Shiny web application.

env
: Standard shiny argument for a render function.

quoted
: Standard shiny argument for a render function.

Examples

```
# See the Shiny vignette in this package for an example.
```

`TableCell` *R6 class that represents a cell in a table.*

Description

The 'TableCell' class represents a cell in a table. Both header cells and body cells are represented by this class.

Format

`R6Class` object.

Active bindings

`instanceId` An integer value that uniquely identifies this cell. NB: This number is guaranteed to be unique within the table, but the method of generation of the values may change in future, so you are advised not to base any logic on specific values.

`cellType` One of the following values that specifies the type of cell: root, rowHeader, columnHeader, cell, total. The cellType controls the default styling that is applied to the cell.

`rowNumber` The row number of the cell. 1 = the first (i.e. top) data row.

`columnNumber` The column number of the cell. 1 = the first (i.e. leftmost) data column.

`visible` TRUE or FALSE to specify whether the cell is rendered.

`rawValue` The original unformatted value.

`formattedValue` The formatted value (i.e. normally of character data type).

`asNBSP` TRUE or FALSE to specify whether cells with no formatted value be output as html nbsp.

`fValueOrNBSP` For internal use by the renderers only.

`isMerged` For internal use by the renderers only.

`isMergeRoot` For internal use by the renderers only.

`mergeIndex` For internal use by the renderers only.

`baseStyleName` The name of the style applied to this cell (a character value). The style must exist in the TableStyles object associated with the table.

`style` A TableStyle object that can apply overrides to the base style for this cell.

Methods

Public methods:

- TableCell$new()
- TableCell$updatePosition()
- TableCell$getCopy()
- TableCell$asList()
- TableCell$asJSON()
- TableCell$clone()

Method `new()`**:** Create a new 'TableCell' object.

Usage:

```
TableCell$new(
  parentTable,
  rowNumber = NULL,
  columnNumber = NULL,
  cellType = "cell",
  visible = TRUE,
  rawValue = NULL,
  formattedValue = NULL,
  baseStyleName = NULL,
  styleDeclarations = NULL,
  asNBSP = FALSE
)
```

Arguments:

`parentTable` Owning table.

`rowNumber` The row number of the cell. 1 = the first (i.e. top) data row.

`columnNumber` The column number of the cell. 1 = the first (i.e. leftmost) data column.

`cellType` One of the following values that specifies the type of cell: root, rowHeader, columnHeader, cell, total. The cellType controls the default styling that is applied to the cell.

`visible` 'TRUE' or 'FALSE' to specify whether the cell is rendered.

`rawValue` The original unformatted value.

`formattedValue` The formatted value (i.e. normally of character data type).

`baseStyleName` The name of the style applied to this cell (a character value). The style must exist in the TableStyles object associated with the table.

`styleDeclarations` A list containing CSS style declarations.

`asNBSP` 'TRUE' or 'FALSE' to specify whether cells with no formatted value be output as html nbsp.

Returns: No return value.

Method `updatePosition()`**:** Set the cell location in the table. Mainly exists for internal use. Typically used after rows/columns/cells are inserted or deleted (which shifts other cells).

Usage:

```
TableCell$updatePosition(rowNumber = NULL, columnNumber = NULL)
```

Arguments:

`rowNumber` The row number of the cell. 1 = the first (i.e. top) data row.

`columnNumber` The column number of the cell. 1 = the first (i.e. leftmost) data column.

Returns: No return value.

Method `getCopy()`**:** Stub only (ignore).

Usage:

```
TableCell$getCopy()
```

Returns: No return value.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
TableCell$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
TableCell$asJSON()
```

Returns: A JSON representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TableCell$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by using the functions in the table.
# It is not intended to be created by users outside of the table.
library(basictabler)
tbl <- qtbl(data.frame(a=1:2, b=3:4))
cell1 <- tbl$cells$setCell(r=4, c=1, cellType="cell", rawValue=5)
cell2 <- tbl$cells$setCell(r=4, c=2, cellType="cell", rawValue=6)
tbl$renderTable()
```

`TableCellRanges` *R6 class that manages cell ranges (e.g. for merged cells).*

Description

The 'TableCellRanges' class contains a list of cell ranges and provides basic utility methods such as finding intersecting ranges to support the functioning of the 'BasicTable' class.

Format

R6Class object.

Active bindings

ranges A list of cell ranges - where each element in the list is another list containing the range extent.

Methods

Public methods:

- TableCellRanges$new()
- TableCellRanges$addRange()
- TableCellRanges$findIntersectingRange()
- TableCellRanges$deleteRange()
- TableCellRanges$clear()
- TableCellRanges$updateAfterRowInsert()
- TableCellRanges$updateAfterRowDelete()
- TableCellRanges$updateAfterColumnInsert()
- TableCellRanges$updateAfterColumnDelete()
- TableCellRanges$asList()
- TableCellRanges$asJSON()
- TableCellRanges$clone()

Method new(): Create a new 'TableCellRanges' object.

Usage:

```
TableCellRanges$new(parentTable)
```

Arguments:

parentTable Owning table.

Returns: No return value.

Method addRange(): Add a cell range to the list of cell ranges. It is not necessary to specify all parameters. rFrom and cFrom must be specified. Only one of rSpan and rTo needs to be specified. Only one of cSpan and cTo needs to be specified.

Usage:

```
TableCellRanges$addRange(
  rFrom = NULL,
  cFrom = NULL,
  rSpan = NULL,
  cSpan = NULL,
  rTo = NULL,
  cTo = NULL
)
```

Arguments:

`rFrom` Row number of the top-left cell in the cell range.

`cFrom` Column number of the top-left cell in the cell range.

`rSpan` Number of rows spanned by the cell range.

`cSpan` Number of columns spanned by the cell range.

`rTo` Row number of the bottom-right cell in the cell range.

`cTo` Column number of the bottom-right cell in the cell range.

Returns: No return value.

Method `findIntersectingRange()`**:** Find a cell range in the list of cell ranges that intersects with the specified cell range. It is not necessary to specify all parameters. rFrom and cFrom must be specified. Only one of rSpan and rTo needs to be specified. Only one of cSpan and cTo needs to be specified.

Usage:

```
TableCellRanges$findIntersectingRange(
  rFrom = NULL,
  cFrom = NULL,
  rSpan = NULL,
  cSpan = NULL,
  rTo = NULL,
  cTo = NULL
)
```

Arguments:

`rFrom` Row number of the top-left cell in the cell range.

`cFrom` Column number of the top-left cell in the cell range.

`rSpan` Number of rows spanned by the cell range.

`cSpan` Number of columns spanned by the cell range.

`rTo` Row number of the bottom-right cell in the cell range.

`cTo` Column number of the bottom-right cell in the cell range.

Returns: No return value.

Method `deleteRange()`**:** Delete the cell range from the list that contains the specified cell.

Usage:

```
TableCellRanges$deleteRange(r = NULL, c = NULL)
```

Arguments:

`r` Row number of a cell in the cell range to be deleted.

`c` Column number of a cell in the cell range to be deleted.

Returns: No return value.

Method `clear()`**:** Clear the list of cell ranges.

Usage:

```
TableCellRanges$clear()
```

Returns: No return value.

Method `updateAfterRowInsert()`**:** Internal use only.

Usage:

```
TableCellRanges$updateAfterRowInsert(r = NULL)
```

Arguments:

`r` Row number.

Returns: No return value.

Method `updateAfterRowDelete()`**:** Internal use only.

Usage:

```
TableCellRanges$updateAfterRowDelete(r = NULL)
```

Arguments:

`r` Row number.

Returns: No return value.

Method `updateAfterColumnInsert()`**:** Internal use only.

Usage:

```
TableCellRanges$updateAfterColumnInsert(c = NULL)
```

Arguments:

`c` Column number.

Returns: No return value.

Method `updateAfterColumnDelete()`**:** Internal use only.

Usage:

```
TableCellRanges$updateAfterColumnDelete(c = NULL)
```

Arguments:

`c` Column number.

Returns: No return value.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
TableCellRanges$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
TableCellRanges$asJSON()
```

Returns: A JSON representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TableCellRanges$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# TableCellRanges objects are never created outside of the BasicTable class.
# For examples of working with merged cells, see the Introduction vignette.
```

TableCells *R6 class that manages cells in a table.*

Description

The 'TableCells' manages the 'TableCell' objects that comprise a 'BasicTable' object.

Format

R6Class object.

Active bindings

`rowCount` The number of rows in the table.

`columnCount` The number of columns in the table.

`rows` The rows of cells in the table - represented as a list, each element of which is a list of 'TableCell' objects.

`all` A list of the cells in the table. Each element in this list is a 'TableCell' object.

Methods

Public methods:

- TableCells$new()
- TableCells$reset()
- TableCells$getCell()
- TableCells$getValue()
- TableCells$getRowValues()
- TableCells$getColumnValues()
- TableCells$setCell()
- TableCells$setBlankCell()
- TableCells$deleteCell()
- TableCells$setValue()
- TableCells$setRow()
- TableCells$setColumn()
- TableCells$extendCells()
- TableCells$moveCell()
- TableCells$insertRow()
- TableCells$deleteRow()
- TableCells$insertColumn()
- TableCells$deleteColumn()
- TableCells$getCells()
- TableCells$findCells()
- TableCells$getColumnWidths()

- TableCells$asList()
- TableCells$asJSON()
- TableCells$clone()

Method `new()`**:** Create a new 'TableCells' object.

Usage:

```
TableCells$new(parentTable = NULL)
```

Arguments:

`parentTable` Owning table.

Returns: No return value.

Method `reset()`**:** Clear all cells from the table.

Usage:

```
TableCells$reset()
```

Returns: No return value.

Method `getCell()`**:** Retrieve a specific 'TableCell' object at the specified location in the table.

Usage:

```
TableCells$getCell(r = NULL, c = NULL)
```

Arguments:

`r` The row number of the cell to retrieve.

`c` The column number of the cell to retrieve.

Returns: A 'TableCell' object.

Method `getValue()`**:** Retrieve the value of a specific 'TableCell' object at the specified location in the table.

Usage:

```
TableCells$getValue(r = NULL, c = NULL, formattedValue = FALSE)
```

Arguments:

`r` The row number of the cell value to retrieve.

`c` The column number of the cell value to retrieve.

`formattedValue` 'TRUE' to retrieve the formatted (character) cell value, 'FALSE' (default) to retrieve the raw cell value (typically numeric).

Returns: The value of the cell.

Method `getRowValues()`**:** Get a vector or list of the values in a row for the entire row or a subset of columns.

Usage:

```
TableCells$getRowValues(
  rowNumber = NULL,
  columnNumbers = NULL,
  formattedValue = FALSE,
  asList = FALSE,
  rebase = TRUE
)
```

Arguments:

`rowNumber` The row number to retrieve the values for (a single row number).

`columnNumbers` The column numbers of the cell value to retrieve (can be a vector of column numbers).

`formattedValue` 'TRUE' to retrieve the formatted (character) cell value, 'FALSE' (default) to retrieve the raw cell value (typically numeric).

`asList` 'TRUE' to retrieve the values as a list, 'FALSE' (default) to retrieve the values as a vector.

`rebase` 'TRUE' to rebase the list/vector so that the first element is at index 1, 'FALSE' to retain the original column numbers.

Returns: A vector or list of the cell values.

Method `getColumnValues()`**:** Get a vector or list of the values in a column for the entire column or a subset of rows.

Usage:

```
TableCells$getColumnValues(
  columnNumber = NULL,
  rowNumbers = NULL,
  formattedValue = FALSE,
  asList = FALSE,
  rebase = TRUE
)
```

Arguments:

`columnNumber` The column number to retrieve the values for (a single column number).

`rowNumbers` The row numbers of the cell value to retrieve (can be a vector of row numbers).

`formattedValue` 'TRUE' to retrieve the formatted (character) cell value, 'FALSE' (default) to retrieve the raw cell value (typically numeric).

`asList` 'TRUE' to retrieve the values as a list, 'FALSE' (default) to retrieve the values as a vector.

`rebase` 'TRUE' to rebase the list/vector so that the first element is at index 1, 'FALSE' to retain the original row numbers.

Returns: A vector or list of the cell values.

Method `setCell()`**:** Create a cell in the table and set the details of the cell.

Usage:

```
TableCells$setCell(
  r = NULL,
  c = NULL,
  cellType = "cell",
  rawValue = NULL,
  formattedValue = NULL,
  visible = TRUE,
  baseStyleName = NULL,
  styleDeclarations = NULL,
  rowSpan = NULL,
  colSpan = NULL
)
```

Arguments:

`r` The row number of the cell.

`c` The column number of the cell.

`cellType` The type of the cell - must be one of the following values: root, rowHeader, columnHeader, cell, total.

`rawValue` The raw value of the cell - typically a numeric value.

`formattedValue` The formatted value of the cell - typically a character value.

`visible` 'TRUE' (default) to specify that the cell is visible, 'FALSE' to specify that the cell will be invisible.

`baseStyleName` The name of a style from the table theme that will be used to style this cell.

`styleDeclarations` A list of CSS style definitions.

`rowSpan` A number greater than 1 to indicate that this cell is merged with cells below. 'NULL' (default) or 1 means the cell is not merged across rows.

`colSpan` A number greater than 1 to indicate that this cell is merged with cells to the right. 'NULL' (default) or 1 means the cell is not merged across columns.

Returns: A vector or list of the cell values.

Method `setBlankCell()`**:** Create an empty cell in the table and set the details of the cell.

Usage:

```
TableCells$setBlankCell(
  r = NULL,
  c = NULL,
  cellType = "cell",
  visible = TRUE,
  baseStyleName = NULL,
  styleDeclarations = NULL,
  rowSpan = NULL,
  colSpan = NULL,
  asNBSP = FALSE
)
```

Arguments:

`r` The row number of the cell.

`c` The column number of the cell.

`cellType` The type of the cell - must be one of the following values: root, rowHeader, columnHeader, cell, total.

`visible` 'TRUE' (default) to specify that the cell is visible, 'FALSE' to specify that the cell will be invisible.

`baseStyleName` The name of a style from the table theme that will be used to style this cell.

`styleDeclarations` A list of CSS style definitions.

`rowSpan` A number greater than 1 to indicate that this cell is merged with cells below. 'NULL' (default) or 1 means the cell is not merged across rows.

`colSpan` A number greater than 1 to indicate that this cell is merged with cells to the right. 'NULL' (default) or 1 means the cell is not merged across columns.

`asNBSP` 'TRUE' if the cell should be rendered as   in HTML, 'FALSE' (default) otherwise.

Returns: A vector or list of the cell values.

Method `deleteCell()`**:** Replace the 'TableCell' object at the specified location in the table with a blank cell.

Usage:

```
TableCells$deleteCell(r = NULL, c = NULL)
```

Arguments:

`r` The row number of the cell value to delete

`c` The column number of the cell value to delete

Returns: The 'TableCell' object that is the new blank cell.

Method `setValue()`**:** Update the value of a cell in the table.

Usage:

```
TableCells$setValue(r = NULL, c = NULL, rawValue = NULL, formattedValue = NULL)
```

Arguments:

`r` The row number of the cell.

`c` The column number of the cell.

`rawValue` The raw value of the cell - typically a numeric value.

`formattedValue` The formatted value of the cell - typically a character value.

Returns: No return value.

Method `setRow()`**:** Create multiple cells in one row of a table.

Usage:

```
TableCells$setRow(
  rowNumber = NULL,
  startAtColumnNumber = 1,
  cellTypes = "cell",
  rawValues = NULL,
  formattedValues = NULL,
  formats = NULL,
  visiblity = TRUE,
  baseStyleNames = NULL,
  fmtFuncArgs = NULL
)
```

Arguments:

`rowNumber` The row number where the cells will be created.

`startAtColumnNumber` The column number to start generating cells at. Default value 1.

`cellTypes` The types of the cells - either a single value or a vector of the same length as rawValues. Each cellType must be one of the following values: root, rowHeader, columnHeader, cell, total.

`rawValues` A vector or list of values. A cell will be generated in the table for each element in the vector/list.

`formattedValues` A vector or list of formatted values. Must be either 'NULL', a single value or a vector/list of the same length as rawValues.

`formats` A vector or list of formats. Must be either 'NULL', a single value or a vector/list of the same length as rawValues.

`visiblity` A logical vector. Must be either a single logical value or a vector/list of the same length as rawValues.

`baseStyleNames` A character vector. Must be either a single style name (from the table theme) or a vector of style names of the same length as rawValues.

`fmtFuncArgs` A list that is length 1 or the same length as the number of columns in the row, where each list element specifies a list of arguments to pass to custom R format functions.

Returns: No return value.

Method `setColumn()`**:** Create multiple cells in one column of a table.

Usage:

```
TableCells$setColumn(
  columnNumber = NULL,
  startAtRowNumber = 2,
  cellTypes = "cell",
  rawValues = NULL,
  formattedValues = NULL,
  formats = NULL,
  visiblity = TRUE,
  baseStyleNames = NULL,
  fmtFuncArgs = NULL
)
```

Arguments:

`columnNumber` The column number where the cells will be created.

`startAtRowNumber` The row number to start generating cells at. Default value 2.

`cellTypes` The types of the cells - either a single value or a vector of the same length as rawValues. Each cellType must be one of the following values: root, rowHeader, columnHeader, cell, total.

`rawValues` A vector or list of values. A cell will be generated in the table for each element in the vector/list.

`formattedValues` A vector or list of formatted values. Must be either 'NULL', a single value or a vector of the same length as rawValues.

`formats` A vector or list of formats. Must be either 'NULL', a single value or a vector of the same length as rawValues.

`visiblity` A logical vector. Must be either a single logical value or a vector of the same length as rawValues.

`baseStyleNames` A character vector. Must be either a single style name (from the table theme) or a vector of style names of the same length as rawValues.

`fmtFuncArgs` A list that is length 1 or the same length as the number of rows in the column, where each list element specifies a list of arguments to pass to custom R format functions.

Returns: No return value.

Method `extendCells()`**:** Enlarge a table to the specified size.

Usage:

```
TableCells$extendCells(rowCount = NULL, columnCount = NULL)
```

Arguments:

`rowCount` The number of rows in the enlarged table.

`columnCount` The number of columns in the enlarged table.

Returns: No return value.

Method `moveCell()`**:** Move a table cell to a different location in the table.

Usage:

```
TableCells$moveCell(r = NULL, c = NULL, cell = NULL)
```

Arguments:

`r` The new row number to move the cell to.

`c` The new column number to move the cell to.

`cell` The 'TableCell' object to move.

Returns: No return value.

Method `insertRow()`**:** Insert a new row in the table at the specified row number and shift existing cells on/below this row down by one row.

Usage:

```
TableCells$insertRow(
  rowNumber = NULL,
  insertBlankCells = TRUE,
  headerCells = 1,
  totalCells = 0
)
```

Arguments:

`rowNumber` The row number where the new row is to be inserted.

`insertBlankCells` 'TRUE' (default) to insert blank cells in the new row, 'FALSE' to create no cells in the new row.

`headerCells` The number of header cells to create at the start of the row. Default value 1.

`totalCells` The number of total cells to create at the end of the row. Default value 0.

Returns: No return value.

Method `deleteRow()`**:** Delete the row in the table at the specified row number and shift existing cells below this row up by one row.

Usage:

```
TableCells$deleteRow(rowNumber = NULL)
```

Arguments:

`rowNumber` The row number of the row to be deleted.

Returns: No return value.

Method `insertColumn()`**:** Insert a new column in the table at the specified column number and shift existing cells in/to the right of this column across by one row.

Usage:

```
TableCells$insertColumn(
  columnNumber = NULL,
  insertBlankCells = TRUE,
  headerCells = 1,
  totalCells = 0
)
```

Arguments:

`columnNumber` The column number where the new column is to be inserted.

`insertBlankCells` 'TRUE' (default) to insert blank cells in the new column, 'FALSE' to create no cells in the new column

`headerCells` The number of header cells to create at the top of the column. Default value 1.

`totalCells` The number of total cells to create at the bottom of the column. Default value 0.

Returns: No return value.

Method `deleteColumn()`**:** Delete the column in the table at the specified column number and shift existing cells to the right of this column to the left by one column.

Usage:

```
TableCells$deleteColumn(columnNumber = NULL)
```

Arguments:

`columnNumber` The column number of the column to be deleted.

Returns: No return value.

Method `getCells()`**:** Retrieve cells by a combination of row and/or column numbers. See the "Finding and Formatting" vignette for graphical examples.

Usage:

```
TableCells$getCells(
  specifyCellsAsList = TRUE,
  rowNumbers = NULL,
  columnNumbers = NULL,
  cellCoordinates = NULL,
  excludeEmptyCells = FALSE,
  matchMode = "simple"
)
```

Arguments:

`specifyCellsAsList` 'TRUE'/'FALSE' to specify how cells are retrieved. Default 'TRUE'. More information is provided in the details section.

`rowNumbers` A vector of row numbers that specify the rows or cells to retrieve.

`columnNumbers` A vector of row numbers that specify the columns or cells to retrieve.

`cellCoordinates` A list of two-element vectors that specify the coordinates of cells to retrieve. Ignored when 'specifyCellsAsList=FALSE'.

`excludeEmptyCells` Default 'FALSE'. Specify 'TRUE' to exclude empty cells.

`matchMode` Either "simple" (default) or "combinations"
"simple" specifies that row and column arguments are considered separately (logical OR), e.g. rowNumbers=1 and columnNumbers=2 will match all cells in row 1 and all cells in

column 2.
"combinations" specifies that row and column arguments are considered together (logical AND), e.g. rowNumbers=1 and columnNumbers=2 will match only the cell single at location (1, 2).
Arguments ‘rowNumbers‘, ‘columnNumbers‘, ‘rowGroups‘ and ‘columnGroups‘ are affected by the match mode. All other arguments are not.

Details: When ‘specifyCellsAsList=TRUE‘ (the default):
Get one or more rows by specifying the row numbers as a vector as the rowNumbers argument and leaving the columnNumbers argument set to the default value of ‘NULL‘, or
Get one or more columns by specifying the column numbers as a vector as the columnNumbers argument and leaving the rowNumbers argument set to the default value of ‘NULL‘, or
Get one or more individual cells by specifying the cellCoordinates argument as a list of vectors of length 2, where each element in the list is the row and column number of one cell,
e.g. ‘list(c(1, 2), c(3, 4))‘ specifies two cells, the first located at row 1, column 2 and the second located at row 3, column 4.
When ‘specifyCellsAsList=FALSE‘:
Get one or more rows by specifying the row numbers as a vector as the rowNumbers argument and leaving the columnNumbers argument set to the default value of ‘NULL‘, or
Get one or more columns by specifying the column numbers as a vector as the columnNumbers argument and leaving the rowNumbers argument set to the default value of ‘NULL‘, or
Get one or more cells by specifying the row and column numbers as vectors for the rowNumbers and columnNumbers arguments, or
a mixture of the above, where for entire rows/columns the element in the other vector is set to ‘NA‘, e.g. to retrieve whole rows, specify the row numbers as the rowNumbers but set the corresponding elements in the columnNumbers vector to ‘NA‘.

Returns: A list of ‘TableCell‘ objects.

Method `findCells()`**:** Find cells matching specified criteria. See the "Finding and Formatting" vignette for graphical examples.

Usage:

```
TableCells$findCells(
  minValue = NULL,
  maxValue = NULL,
  exactValues = NULL,
  valueRanges = NULL,
  includeNull = TRUE,
  includeNA = TRUE,
  emptyCells = "include",
  rowNumbers = NULL,
  columnNumbers = NULL,
  cellCoordinates = NULL,
  cells = NULL,
  rowColumnMatchMode = "simple"
)
```

Arguments:

`minValue` A numerical value specifying a minimum value threshold.

`maxValue` A numerical value specifying a maximum value threshold.

`exactValues` A vector or list specifying a set of allowed values.

`valueRanges` A vector specifying one or more value range expressions which the cell values must match. If multiple value range expressions are specified, then the cell value must match any of one the specified expressions.

`includeNull` Specify TRUE to include 'NULL' in the matched cells, FALSE to exclude 'NULL' values.

`includeNA` Specify TRUE to include 'NA' in the matched cells, FALSE to exclude 'NA' values.

`emptyCells` A word that specifies how empty cells are matched - must be one of "include" (default), "exclude" or "only".

`rowNumbers` A vector of row numbers that specify the rows or cells to constrain the search.

`columnNumbers` A vector of column numbers that specify the columns or cells to constrain the search.

`cellCoordinates` A list of two-element vectors that specify the coordinates of cells to constrain the search.

`cells` A 'TableCell' object or a list of 'TableCell' objects to constrain the scope of the search.

`rowColumnMatchMode` Either "simple" (default) or "combinations":
"simple" specifies that row and column arguments are considered separately (logical OR), e.g. rowNumbers=1 and columnNumbers=2 will match all cells in row 1 and all cells in column 2.
"combinations" specifies that row and column arguments are considered together (logical AND), e.g. rowNumbers=1 and columnNumbers=2 will match only the cell single at location (1, 2).
Arguments 'rowNumbers', 'columnNumbers', 'rowGroups' and 'columnGroups' are affected by the match mode. All other arguments are not.

Returns: A list of 'TableCell' objects.

Method `getColumnWidths()`**:** Retrieve the width of the longest value (in characters) in each column.

Usage:

```
TableCells$getColumnWidths()
```

Returns: The width of the column in characters.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
TableCells$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
TableCells$asJSON()
```

Returns: A JSON representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TableCells$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
# This class should only be created by the table.
# It is not intended to be created outside of the table.
library(basictabler)
tbl <- qtbl(data.frame(a=1:2, b=3:4))
cells <- tbl$cells
cells$setCell(r=4, c=1, cellType="cell", rawValue=5)
cells$setCell(r=4, c=2, cellType="cell", rawValue=6)
tbl$renderTable()
```

TableHtmlRenderer *R6 class that renders a table in HTML.*

Description

The 'TableHtmlRenderer' class creates a HTML representation of a table.

Format

R6Class object.

Methods

Public methods:

- TableHtmlRenderer$new()
- TableHtmlRenderer$getTableHtml()
- TableHtmlRenderer$clone()

Method `new()`**:** Create a new 'TableHtmlRenderer' object.

Usage:

```
TableHtmlRenderer$new(parentTable)
```

Arguments:

parentTable Owning table.

Returns: No return value.

Method `getTableHtml()`**:** Generate a HTML representation of the table.

Usage:

```
TableHtmlRenderer$getTableHtml(styleNamePrefix = NULL)
```

Arguments:

`styleNamePrefix` A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple tables exist.

Returns: A list containing HTML tags from the 'htmltools' package. Convert this to a character variable using 'as.character()'.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TableHtmlRenderer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class is for internal use only.  It is
# created only by the BasicTable class when rendering to HTML.
# See the package vignettes for more information about outputs.
library(basictabler)
tbl <- qtbl(data.frame(a=1:2, b=3:4))
tbl$renderTable()
```

`TableOpenXlsxRenderer` *R6 class that renders a table into an Excel worksheet.*

Description

The 'TableOpenXlsxRenderer' class creates a representation of a table in an Excel file using the 'openxlsx' package. See the Excel Output vignette for more details.

Format

`R6Class` object.

Methods

Public methods:

- `TableOpenXlsxRenderer$new()`
- `TableOpenXlsxRenderer$writeToCell()`
- `TableOpenXlsxRenderer$writeToWorksheet()`
- `TableOpenXlsxRenderer$clone()`

Method `new()`**:** Create a new 'TableOpenXlsxRenderer' object.

Usage:

```
TableOpenXlsxRenderer$new(parentTable)
```

Arguments:

`parentTable` Owning table.

Returns: No return value.

Method `writeToCell()`**:** Write a value to a cell, optionally with styling and cell merging.

Usage:

```
TableOpenXlsxRenderer$writeToCell(
  wb = NULL,
  wsName = NULL,
  rowNumber = NULL,
  columnNumber = NULL,
  value = NULL,
  applyStyles = TRUE,
  baseStyleName = NULL,
  style = NULL,
  mapFromCss = TRUE,
  mergeRows = NULL,
  mergeColumns = NULL
)
```

Arguments:

`wb` A workbook object from the openxlsx package.

`wsName` The name of the worksheet where the value is to be written.

`rowNumber` The row number of the cell where the value is to be written.

`columnNumber` The column number of the cell where the value is to be written.

`value` The value to be written.

`applyStyles` 'TRUE' (default) to also set the styling of the cell, 'FALSE' to only write the value.

`baseStyleName` The name of the style from the table theme to apply to the cell.

`style` A 'TableStyle' object that contains additional styling to apply to the cell.

`mapFromCss` 'TRUE' (default) to map the basictabler CSS styles to corresponding Excel styles, 'FALSE' to apply only the specified xl styles.

`mergeRows` If the cell is to be merged with adjacent cells, then an integer or numeric vector specifying the row numbers of the merged cell. NULL (default) to not merge cells.

`mergeColumns` If the cell is to be merged with adjacent cells, then an integer or numeric vector specifying the column numbers of the merged cell. NULL (default) to not merge cells.

Returns: No return value.

Method `writeToWorksheet()`**:** Write a table to an Excel worksheet.

Usage:

```
TableOpenXlsxRenderer$writeToWorksheet(
  wb = NULL,
  wsName = NULL,
  topRowNumber = NULL,
  leftMostColumnNumber = NULL,
  outputValuesAs = "rawValue",
  useFormattedValueIfRawValueIsNull = TRUE,
  applyStyles = TRUE,
  mapStylesFromCSS = TRUE
)
```

Arguments:

`wb` A workbook object from the openxlsx package.

`wsName` The name of the worksheet where the value is to be written.

`topRowNumber` The row number of the top-left cell where the table is to be written.

`leftMostColumnNumber` The column number of the top-left cell where the table is to be written.

`outputValuesAs` Specify whether the raw or formatted values should be written to the worksheet. Value must be one of "rawValue", "formattedValueAsText", "formattedValueAsNumber".

`useFormattedValueIfRawValueIsNull` 'TRUE' to use the formatted cell value instead of the raw cell value if the raw value is 'NULL'. 'FALSE' to always use the raw value. Default 'TRUE'.

`applyStyles` 'TRUE' (default) to also set the styling of the cells, 'FALSE' to only write the value.

`mapStylesFromCSS` 'TRUE' (default) to map the basictabler CSS styles to corresponding Excel styles, 'FALSE' to apply only the specified xl styles.

Returns: No return value.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TableOpenXlsxRenderer$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class is for internal use only.  It is
# created only by the BasicTable class when rendering to Excel.
library(basictabler)
tbl <- qtbl(data.frame(a=1:2, b=3:4))
library(openxlsx)
wb <- createWorkbook(creator = Sys.getenv("USERNAME"))
addWorksheet(wb, "Data")
tbl$writeToExcelWorksheet(wb=wb, wsName="Data",
                          topRowNumber=1, leftMostColumnNumber=1,
                          applyStyles=TRUE, mapStylesFromCSS=TRUE)
# Use saveWorkbook() to save the Excel file.
```

`TableOpenXlsxStyle` *R6 class that specifies Excel styling as used by the openxlsx package.*

Description

The 'TableOpenXlsxStyle' class specifies the styling for cells in an Excel worksheet.

Format

R6Class object.

Active bindings

`baseStyleName` The name of the base style in the table.

`isBaseStyle` 'TRUE' when this style is the equivalent of a named style in the table, 'FALSE' if this style has additional settings over and above the base style of the same name.

`fontName` The name of the font (single font name, not a CSS style list).

`fontSize` The size of the font (units: point).

`bold` TRUE' if text is bold.

`italic` 'TRUE' if text is italic.

`underline` 'TRUE' if text is underlined.

`strikethrough` 'TRUE' if text has a line through it.

`superscript` 'TRUE' if text is small and raised.

`subscript` 'TRUE' if text is small and lowered.

`fillColor` The background colour for the cell (as a hex value, e.g. #00FF00).

`textColor` The color of the text (as a hex value).

`hAlign` The horizontal alignment of the text: left, center or right.

`vAlign` The vertical alignment of the text: top, middle or bottom.

`wrapText` TRUE if the text is allowed to wrap onto multiple lines.

`textRotation` The rotation angle of the text or 255 for vertical.

`indent` The text indentation.

`borderAll` A list (with elements style and color) specifying the border settings for all four sides of each cell at once.

`borderLeft` A list (with elements style and color) specifying the border settings for the left border of each cell.

`borderRight` A list (with elements style and color) specifying the border settings for the right border of each cell.

`borderTop` A list (with elements style and color) specifying the border settings for the top border of each cell.

`borderBottom` A list (with elements style and color) specifying the border settings for the bottom border of each cell.

`valueFormat` The Excel formatting applied to the field value. One of the following values: "GENERAL", "NUMBER", "CURRENCY", "ACCOUNTING", "DATE", "LONGDATE", TIME, "PERCENTAGE", "FRACTION", "SCIENTIFIC", "TEXT", "COMMA". Or for dates/datetimes, a combination of d, m, y. Or for numeric values, use 0.00 etc.

`minColumnWidth` The minimum width of this column.

`minRowHeight` The minimum height of this row.

`openxlsxStyle` The return value from openxlsx::createStyle().

Methods

Public methods:

- TableOpenXlsxStyle$new()
- TableOpenXlsxStyle$isBasicStyleNameMatch()
- TableOpenXlsxStyle$isFullStyleDetailMatch()
- TableOpenXlsxStyle$createOpenXslxStyle()
- TableOpenXlsxStyle$asList()
- TableOpenXlsxStyle$asJSON()
- TableOpenXlsxStyle$asString()
- TableOpenXlsxStyle$clone()

Method `new()`**:** Create a new 'TableOpenXlsxStyle' object.

Usage:

```
TableOpenXlsxStyle$new(
  parentTable,
  baseStyleName = NULL,
  isBaseStyle = NULL,
  fontName = NULL,
  fontSize = NULL,
  bold = NULL,
  italic = NULL,
  underline = NULL,
  strikethrough = NULL,
  superscript = NULL,
  subscript = NULL,
  fillColor = NULL,
  textColor = NULL,
  hAlign = NULL,
  vAlign = NULL,
  wrapText = NULL,
  textRotation = NULL,
  indent = NULL,
  borderAll = NULL,
  borderLeft = NULL,
  borderRight = NULL,
  borderTop = NULL,
  borderBottom = NULL,
  valueFormat = NULL,
  minColumnWidth = NULL,
  minRowHeight = NULL
)
```

Arguments:

`parentTable` Owning table.

`baseStyleName` The name of the base style in the table.

`isBaseStyle` 'TRUE' when this style is the equivalent of a named style in the table, 'FALSE' if this style has additional settings over and above the base style of the same name.

`fontName` The name of the font (single font name, not a CSS style list).

`fontSize` The size of the font (units: point).

`bold` 'TRUE' if text is bold.

`italic` 'TRUE' if text is italic.

`underline` 'TRUE' if text is underlined.

`strikethrough` 'TRUE' if text has a line through it.

`superscript` 'TRUE' if text is small and raised.

`subscript` 'TRUE' if text is small and lowered.

`fillColor` The background colour for the cell (as a hex value, e.g. #00FF00).

`textColor` The color of the text (as a hex value).

`hAlign` The horizontal alignment of the text: left, center or right.

`vAlign` The vertical alignment of the text: top, middle or bottom.

`wrapText` 'TRUE' if the text is allowed to wrap onto multiple lines.

`textRotation` The rotation angle of the text or 255 for vertical.

`indent` The text indentation.

`borderAll` A list (with elements style and color) specifying the border settings for all four sides of each cell at once.

`borderLeft` A list (with elements style and color) specifying the border settings for the left border of each cell.

`borderRight` A list (with elements style and color) specifying the border settings for the right border of each cell.

`borderTop` A list (with elements style and color) specifying the border settings for the top border of each cell.

`borderBottom` A list (with elements style and color) specifying the border settings for the bottom border of each cell.

`valueFormat` The Excel formatting applied to the field value. One of the following values: "GENERAL", "NUMBER", "CURRENCY", "ACCOUNTING", "DATE", "LONGDATE", TIME, "PERCENTAGE", "FRACTION", "SCIENTIFIC", "TEXT", "COMMA". Or for dates/datetimes, a combination of d, m, y. Or for numeric values, use 0.00 etc.

`minColumnWidth` The minimum width of this column.

`minRowHeight` The minimum height of this row.

Returns: No return value.

Method `isBasicStyleNameMatch()`**:** Check if this style matches the specified base style name.

Usage:

```
TableOpenXlsxStyle$isBasicStyleNameMatch(baseStyleName = NULL)
```

Arguments:

`baseStyleName` The style name to compare to.

Returns: 'TRUE' if the style name matches, 'FALSE' otherwise.

Method `isFullStyleDetailMatch()`**:** Check if this style matches the specified style properties.

Usage:

```
TableOpenXlsxStyle$isFullStyleDetailMatch(
  baseStyleName = NULL,
  isBaseStyle = NULL,
  fontName = NULL,
  fontSize = NULL,
  bold = NULL,
  italic = NULL,
  underline = NULL,
  strikethrough = NULL,
  superscript = NULL,
  subscript = NULL,
  fillColor = NULL,
  textColor = NULL,
  hAlign = NULL,
  vAlign = NULL,
  wrapText = NULL,
  textRotation = NULL,
  indent = NULL,
  borderAll = NULL,
  borderLeft = NULL,
  borderRight = NULL,
  borderTop = NULL,
  borderBottom = NULL,
  valueFormat = NULL,
  minColumnWidth = NULL,
  minRowHeight = NULL
)
```

Arguments:

`baseStyleName` The style name to compare to.

`isBaseStyle` Whether the style being compared to is a base style.

`fontName` The font name to compare to.

`fontSize` The font size to compare to.

`bold` The style property bold to compare to.

`italic` The style property italic to compare to.

`underline` The style property underline to compare to.

`strikethrough` The style property strikethrough to compare to.

`superscript` The style property superscript to compare to.

`subscript` The style property subscript to compare to.

`fillColor` The style property fillColor to compare to.

`textColor` The style property textColor to compare to.

`hAlign` The style property hAlign to compare to.

`vAlign` The style property vAlign to compare to.

`wrapText` The style property wrapText to compare to.

`textRotation` The style property textRotation to compare to.

`indent` The style property indent to compare to.

`borderAll` The style property borderAll to compare to.

`borderLeft` The style property borderLeft to compare to.

`borderRight` The style property borderRight to compare to.

`borderTop` The style property borderTop to compare to.

`borderBottom` The style property borderBottom to compare to.

`valueFormat` The style value format to compare to.

`minColumnWidth` The style property minColumnWidth to compare to.

`minRowHeight` The style property minRowHeight to compare to.

Returns: 'TRUE' if the style matches, 'FALSE' otherwise.

Method `createOpenXslxStyle()`**:** Create the 'openxlsx' style based on the specified style properties.

Usage:

```
TableOpenXlsxStyle$createOpenXslxStyle()
```

Returns: No return value.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
TableOpenXlsxStyle$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
TableOpenXlsxStyle$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return the contents of this object as a string for debugging.

Usage:

```
TableOpenXlsxStyle$asString()
```

Returns: A character representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TableOpenXlsxStyle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should only be created by using the functions in the table.
# It is not intended to be created by users outside of the table.
library(basictabler)
tbl <- qtbl(data.frame(a=1:2, b=3:4))
library(openxlsx)
```

```
wb <- createWorkbook(creator = Sys.getenv("USERNAME"))
addWorksheet(wb, "Data")
tbl$writeToExcelWorksheet(wb=wb, wsName="Data",
                          topRowNumber=1, leftMostColumnNumber=1,
                          applyStyles=TRUE, mapStylesFromCSS=TRUE)
# Use saveWorkbook() to save the Excel file.
```

`TableOpenXlsxStyles` *R6 class that defines a collection of Excel styles as used by the 'openxlsx' package.*

Description

The 'TableOpenXlsxStyles' class stores a collection of 'TableTableOpenXlsx' style objects.

Format

`R6Class` object.

Active bindings

`count` The number of styles in the collection.

`styles` A list of 'TableOpenXlsxStyle' objects that comprise the collection.

Methods

Public methods:

- `TableOpenXlsxStyles$new()`
- `TableOpenXlsxStyles$clearStyles()`
- `TableOpenXlsxStyles$findNamedStyle()`
- `TableOpenXlsxStyles$findOrAddStyle()`
- `TableOpenXlsxStyles$addNamedStyles()`
- `TableOpenXlsxStyles$asList()`
- `TableOpenXlsxStyles$asJSON()`
- `TableOpenXlsxStyles$asString()`
- `TableOpenXlsxStyles$clone()`

Method `new()`**:** Create a new 'TableOpenXlsxStyles' object.

Usage:

```
TableOpenXlsxStyles$new(parentTable)
```

Arguments:

`parentTable` Owning table.

Returns: No return value.

Method `clearStyles()`**:** Clear the collection removing all styles.

Usage:

```
TableOpenXlsxStyles$clearStyles()
```

Returns: No return value.

Method `findNamedStyle()`**:** Find a style in the collection matching the specified base style name.

Usage:

```
TableOpenXlsxStyles$findNamedStyle(baseStyleName)
```

Arguments:

`baseStyleName` The style name to find.

Returns: A 'TableTableOpenXlsx' object that is the style matching the specified base style name or 'NULL' otherwise.

Method `findOrAddStyle()`**:** Find a style in the collection matching the specified base style name and style properties. If there is no matching style, then optionally add a new style.

Usage:

```
TableOpenXlsxStyles$findOrAddStyle(
  action = "findOrAdd",
  baseStyleName = NULL,
  isBaseStyle = NULL,
  style = NULL,
  mapFromCss = TRUE
)
```

Arguments:

`action` The action to carry out. Must be one of "find", "add" or "findOrAdd" (default).

`baseStyleName` The style name to find/add.

`isBaseStyle` Is the style to be found/added a base style?

`style` A 'TableStyle' object specifying style properties to be found/added.

`mapFromCss` 'TRUE' (default) to map the basictabler CSS styles to corresponding Excel styles, 'FALSE' to apply only the specified xl styles.

Returns: A 'TableTableOpenXlsx' object that is the style matching the specified base style name or 'NULL' otherwise.

Method `addNamedStyles()`**:** Populate the OpenXlsx styles based on the styles defined in the table.

Usage:

```
TableOpenXlsxStyles$addNamedStyles(mapFromCss = TRUE)
```

Arguments:

`mapFromCss` 'TRUE' (default) to map the basictabler CSS styles to corresponding Excel styles, 'FALSE' to apply only the specified xl styles.

Returns: No return value.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
TableOpenXlsxStyles$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
TableOpenXlsxStyles$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return the contents of this object as a string for debugging.

Usage:

```
TableOpenXlsxStyles$asString(seperator = ", ")
```

Arguments:

`seperator` Delimiter used to combine multiple values into a string.

Returns: A character representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TableOpenXlsxStyles$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# This class should not be used by end users.  It is an internal class
# created only by the BasicTable class.  It is used when rendering to Excel.
library(basictabler)
tbl <- qtbl(data.frame(a=1:2, b=3:4))
library(openxlsx)
wb <- createWorkbook(creator = Sys.getenv("USERNAME"))
addWorksheet(wb, "Data")
tbl$writeToExcelWorksheet(wb=wb, wsName="Data",
                          topRowNumber=1, leftMostColumnNumber=1,
                          applyStyles=TRUE, mapStylesFromCSS=TRUE)
# Use saveWorkbook() to save the Excel file.
```

`TableStyle` *R6 class that specifies styling.*

Description

The 'TableStyle' class specifies the styling for headers and cells in a table. Styles are specified in the form of Cascading Style Sheet (CSS) name-value pairs.

Format

R6Class object.

Active bindings

`name` The unique name of the style (must be unique among the style names in the table theme).

`declarations` A list containing CSS style declarations. Example: 'declarations = list(font="...", color="...")'

Methods

Public methods:

- `TableStyle$new()`
- `TableStyle$setPropertyValue()`
- `TableStyle$setPropertyValues()`
- `TableStyle$getPropertyValue()`
- `TableStyle$asCSSRule()`
- `TableStyle$asNamedCSSStyle()`
- `TableStyle$getCopy()`
- `TableStyle$asList()`
- `TableStyle$asJSON()`
- `TableStyle$clone()`

Method `new()`**:** Create a new 'TableStyle' object.

Usage:

```
TableStyle$new(parentTable, styleName = NULL, declarations = NULL)
```

Arguments:

`parentTable` Owning table.

`styleName` A unique name for the style.

`declarations` A list containing CSS style declarations. Example: 'declarations = list(font="...", color="...")'

Returns: No return value.

Method `setPropertyValue()`**:** Set the value of a single style property.

Usage:

```
TableStyle$setPropertyValue(property = NULL, value = NULL)
```

Arguments:

`property` The CSS style property name, e.g. color.

`value` The value of the style property, e.g. red.

Returns: No return value.

Method `setPropertyValues()`**:** Set the values of multiple style properties.

Usage:

```
TableStyle$setPropertyValues(declarations = NULL)
```

Arguments:

`declarations` A list containing CSS style declarations. Example: 'declarations = list(font="...", color="...")'

Returns: No return value.

Method `getPropertyValue()`**:** Get the value of a single style property.

Usage:

```
TableStyle$getPropertyValue(property = NULL)
```

Arguments:

`property` The CSS style property name, e.g. color.

Returns: No return value.

Method `asCSSRule()`**:** Generate a CSS style rule from this table style.

Usage:

```
TableStyle$asCSSRule(selector = NULL)
```

Arguments:

`selector` The CSS selector name. Default value 'NULL'.

Returns: The CSS style rule, e.g. { text-align: center; color: red; }

Method `asNamedCSSStyle()`**:** Generate a named CSS style from this table style.

Usage:

```
TableStyle$asNamedCSSStyle(styleNamePrefix = NULL)
```

Arguments:

`styleNamePrefix` A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple tables exist.

Returns: The CSS style rule, e.g. cell { text-align: center; color: red; }

Method `getCopy()`**:** Create a copy of this 'TableStyle' object.

Usage:

```
TableStyle$getCopy(newStyleName = NULL)
```

Arguments:

`newStyleName` The name of the new style.

Returns: The new 'TableStyle' object.

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
TableStyle$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
TableStyle$asJSON()
```

Returns: A JSON representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TableStyle$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# TableStyle objects are normally created indirectly via one of the helper
# methods.
# For an example, see the `TableStyles` class.
```

`TableStyles` *R6 class that defines a collection of styles.*

Description

The 'TableStyles' class defines all of the base styles needed to style/theme a table. It defines the names of the styles that are used for styling the different parts of the table.

Format

`R6Class` object.

Active bindings

`count` The number of styles in this styles collection.

`theme` The name of the theme.

`styles` The collection of 'TableStyle' objects in this styles collection.

`allowExternalStyles` Enable integration scenarios where an external system is supplying the CSS definitions.

`tableStyle` The name of the style for the HTML table element.

`rootStyle` The name of the style for the HTML cell at the top left of the table (when both row and column headers are displayed).

`rowHeaderStyle` The name of the style for the row headers in the table.

`colHeaderStyle` The name of the style for the column headers in the table.

`cellStyle` The name of the cell style for the non-total cells in the body of the table.

`totalStyle` The name of the cell style for the total cells in the table.

Methods

Public methods:

- TableStyles$new()
- TableStyles$isExistingStyle()
- TableStyles$getStyle()
- TableStyles$addStyle()
- TableStyles$copyStyle()
- TableStyles$asCSSRule()
- TableStyles$asNamedCSSStyle()
- TableStyles$asList()
- TableStyles$asJSON()
- TableStyles$asString()
- TableStyles$clone()

Method `new()`**:** Create a new 'TableStyles' object.

Usage:

```
TableStyles$new(parentTable, themeName = NULL, allowExternalStyles = FALSE)
```

Arguments:

`parentTable` Owning table.

`themeName` The name of the theme.

`allowExternalStyles` Enable integration scenarios where an external system is supplying the CSS definitions.

Returns: No return value.

Method `isExistingStyle()`**:** Check whether a style with the specified name exists in the collection.

Usage:

```
TableStyles$isExistingStyle(styleName = NULL)
```

Arguments:

`styleName` The style name.

Returns: 'TRUE' if a style with the specified name exists, 'FALSE' otherwise.

Method `getStyle()`**:** Retrieve a style with the specified name from the collection.

Usage:

```
TableStyles$getStyle(styleName = NULL)
```

Arguments:

`styleName` The style name.

Returns: A 'TableStyle' object if a style with the specified name exists in the collection, an error is raised otherwise.

Method `addStyle()`**:** Add a new style to the collection of styles.

Usage:

```
TableStyles$addStyle(styleName = NULL, declarations = NULL)
```

Arguments:

`styleName` The style name of the new style.

`declarations` A list containing CSS style declarations. Example: 'declarations = list(font="...", color="...")'

Returns: The newly created 'TableStyle' object.

Method `copyStyle()`**:** Create a copy of an exist style.

Usage:

```
TableStyles$copyStyle(styleName = NULL, newStyleName = NULL)
```

Arguments:

`styleName` The style name of the style to copy.

`newStyleName` The name of the new style.

Returns: The newly created 'TableStyle' object.

Method `asCSSRule()`**:** Generate a CSS style rule from the specified table style.

Usage:

```
TableStyles$asCSSRule(styleName = NULL, selector = NULL)
```

Arguments:

`styleName` The style name.

`selector` The CSS selector name. Default value 'NULL'.

Returns: The CSS style rule, e.g. { text-align: center; color: red; }

Method `asNamedCSSStyle()`**:** Generate a named CSS style from the specified table style.

Usage:

```
TableStyles$asNamedCSSStyle(styleName = NULL, styleNamePrefix = NULL)
```

Arguments:

`styleName` The style name.

`styleNamePrefix` A character variable specifying a prefix for all named CSS styles, to avoid style name collisions where multiple tables exist.

Returns: The CSS style rule, e.g. cell { text-align: center; color: red; }

Method `asList()`**:** Return the contents of this object as a list for debugging.

Usage:

```
TableStyles$asList()
```

Returns: A list of various object properties.

Method `asJSON()`**:** Return the contents of this object as JSON for debugging.

Usage:

```
TableStyles$asJSON()
```

Returns: A JSON representation of various object properties.

Method `asString()`**:** Return the contents of this object as a string for debugging.

Usage:

```
TableStyles$asString(seperator = ", ")
```

Arguments:

`seperator` Delimiter used to combine multiple values into a string.

Returns: A character representation of various object properties.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
TableStyles$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
# Creating styles is part of defining a theme for a table.
# Multiple styles must be created for each theme.
# The example below shows how to create one style.
# For an example of creating a full theme please
# see the Styling vignette.
tbl <- BasicTable$new()
# ...
TableStyles <- TableStyles$new(tbl, themeName="compact")
TableStyles$addStyle(styleName="MyNewStyle", list(
    font="0.75em arial",
    padding="2px",
    border="1px solid lightgray",
    "vertical-align"="middle",
    "text-align"="center",
    "font-weight"="bold",
    "background-color"="#F2F2F2"
  ))
```

---

trainstations *Train Stations*

Description

A reference dataset listing the codes, names and locations of trains stations in Great Britain.

Usage

```
trainstations
```

Format

A data frame with 2568 rows and 7 variables:

CrsCode 3-letter code for the station

StationName The name of the station

OsEasting The UK Ordnance Survey Easting coordinate for the station

OsNorthing The UK Ordnance Survey Northing coordinate for the station

GridReference Grid reference for the station

Latitude Latitude of the station location

Longitude Longitude of the station location

Source

https://www.recenttraintimes.co.uk/

---

vreConvertSimpleNumericRange

Convert a simple range expression to a standard R logical expression.

Description

`vreConvertSimpleNumericRange` is a utility function that converts a simple range expression of the form "value1<=v<value2" to a standard R logical expression of the form "value1<=v && v<value2".

Usage

```
vreConvertSimpleNumericRange(vre)
```

Arguments

`vre`	The value range expression to examine.

Value

A standard R logical expression.

vreGetSingleValue *Read the value from a single-valued value range expression.*

Description

vreGetSingleValue is a utility function reads the single value from a value range expression (it assumes the specified is either numeric, a number expressed as text or an expression of the form "v=" or "v==").

Usage

```
vreGetSingleValue(vre)
```

Arguments

vre	The value range expression to examine.

Value

The value read from the expression.

vreHexToClr *Convert a colour in hex format (#RRGGBB) into a list.*

Description

vreHexToClr converts a colour in hex format (#RRGGBB) into a list of three element (r, g and b).

Usage

```
vreHexToClr(hexclr)
```

Arguments

hexclr	The colour to convert.

Value

The converted colour.

vreIsEqual *Test if two numeric values are equal within tolerance.*

Description

vreIsEqual tests whether two values are equal within sqrt(.Machine$double.eps).

Usage

```
vreIsEqual(value1, value2)
```

Arguments

value1	The first value to compare.
value2	The second value to compare.

Value

'TRUE' if the two numbers are equal, 'FALSE' otherwise.

vreIsMatch *Test whether a value matches a value range expression.*

Description

vreIsMatch tests a value (e.g. from a cell) matches the criteria specified in a value range expression.

Usage

```
vreIsMatch(vre, v, testOnly = FALSE)
```

Arguments

vre	The value range expression.
v	The value.
testOnly	'TRUE' if this comparison is just a test.

Value

'TRUE' if v matches the criteria specified in the value range expression, 'FALSE' otherwise.

vreIsSimpleNumericRange

Determine if a value range expression is a simple range expression.

Description

vreIsSingleValue is a utility function that returns 'TRUE' if the specified value range expression is a simple range expression of the form "value1<=v<value2", where the logical comparisons can be < or <= only and the values must be numbers.

Usage

```
vreIsSimpleNumericRange(vre)
```

Arguments

vre	The value range expression to examine.

Value

'TRUE' if vre is a simple range expression, 'FALSE' otherwise.

vreIsSingleValue

Determine if a value range expression is a single value.

Description

vreIsSingleValue is a utility function that returns 'TRUE' if the specified value range expression is either numeric, a number expressed as text or an expression of the form "v=" or "v==".

Usage

```
vreIsSingleValue(vre)
```

Arguments

vre	The value range expression to examine.

Value

'TRUE' if vre is a single value, 'FALSE' otherwise.

vreScale2Colours *Scale a number from a range into a colour gradient.*

Description

vreScale2Colours takes a value from a range and scales it proportionally into a colour from a colour gradient.

Usage

```
vreScale2Colours(clr1, clr2, vMin, vMax, value)
```

Arguments

clr1	The colour representing the lower value of the target range.
clr2	The colour representing the upper value of the target range.
vMin	The lower value of the source range.
vMax	The upper value of the source range.
value	The source value to rescale into the target range.

Value

The value scaled into the target colour gradient.

vreScaleNumber *Rescale a number from one range into another range.*

Description

vreScaleNumber takes a value from one range and scales it proportionally into another range.

Usage

```
vreScaleNumber(n1, n2, vMin, vMax, value, decimalPlaces = 3)
```

Arguments

n1	The lower value of the target range.
n2	The upper value of the target range.
vMin	The lower value of the source range.
vMax	The upper value of the source range.
value	The source value to rescale into the target range.
decimalPlaces	The number of decimal places to round the result to.

Value

The value rescaled into the target range.

Index