Package 'basictabler'

March 7, 2020

Type Package

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

Version 0.3.1

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), jsonlite (>= 1.1), htmltools(>= 0.3.5), htmlwidgets (>= 0.8)

Suggests lubridate (>= 1.5.0), listviewer (>= 1.4.0), openxlsx (>= 4.0.17), shiny, knitr, rmarkdown, testthat

License GPL-3

Encoding UTF-8

LazyData true

URL https://github.com/cbailiss/basictabler

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

VignetteBuilder knitr

RoxygenNote 7.0.2

NeedsCompilation no

Author Christopher Bailiss [aut, cre]

Maintainer Christopher Bailiss <cbailiss@gmail.com>

Repository CRAN

Date/Publication 2020-03-07 11:00:02 UTC

R topics documented:

BasicTable . . . 3
basictabler . . . 5
basictablerOutput . . . 6
basictablerSample . . . 6
bhmsummary . . . 7
checkArgument . . . 8
cleanCssValue . . . 9
containsText . . . 10
getBlankTblTheme . . . 10
getCompactTblTheme . . . 11
getDefaultTblTheme . . . 11
getLargePlainTblTheme . . . 12
getNextPosition . . . 12
getSimpleColoredTblTheme . . . 13
getTblTheme . . . 14
getXlBorderFromCssBorder . . . 14
getXlBorderStyleFromCssBorder . . . 15
isNumericValue . . . 15
isTextValue . . . 16
oneToNULL . . . 16
parseColor . . . 17
parseCssBorder . . . 17
parseCssSizeToPt . . . 18
parseCssSizeToPx . . . 18
parseCssString . . . 19
parseXlBorder . . . 19
qhtbl . . . 20
qtbl . . . 21
renderBasictabler . . . 23
TableCell . . . 23
TableCellRanges . . . 24
TableCells . . . 25
TableHtmlRenderer . . . 27
TableOpenXlsxRenderer . . . 28
TableOpenXlsxStyle . . . 29
TableOpenXlsxStyles . . . 30
TableStyle . . . 32
TableStyles . . . 33
trainstations . . . 34

Index **35**

BasicTable *A 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.

Value

Object of R6Class with properties and methods that define a basic table.

Fields

`argumentCheckMode` A number (0-4 meaning none, minimal, basic, balanced, full) indicating the argument checking level.

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

`rowCount` The number of rows in the table.

`columnCount` The number of columns in the table.

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

`theme` The name of the theme currently applied to the table.

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

`allowExternalStyles` Enable support for external styles, when producing content for external systems.

`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

Documentation For more complete explanations and examples please see the extensive vignettes supplied with this package.

`new(argumentCheckMode="auto", theme=NULL, replaceExistingStyles=FALSE, tableStyle=NULL, headingStyle=`
Create a new table, optionally specifying the name of a built in theme or CSS style declarations for the different cells within the table.

`addData(dataFrame=NULL, columnNamesAsColumnHeaders=TRUE, explicitColumnHeaders=NULL, rowNamesAsRowHea`
Generate the table from a data frame, specifying headers and value formatting.

`addMatrix(matrix=NULL, columnNamesAsColumnHeaders=TRUE, explicitColumnHeaders=NULL, rowNamesAsRowHead`
Generate the table from a matrix, specifying headers and value formatting.l

`mergeCells(rFrom, cFrom, rSpan=NULL, cSpan=NULL, rTo=NULL, cTo=NULL)` Merge cells in the table. This does not delete the other cells covered by the merged cell. When the table is output, the top-left most cell in the merged cell range is rendered over the other cells (which are effectively hidden).

`unmergeCells(r, c, errorIfNotFound=TRUE)` Delete a merged cell range by specifying any of the cells covered by the merged cell.

`applyCellMerges()` Updates the isMerged, isMergeRoot and mergeIndex properties of the cells.

`formatValue(value=NULL, format=NULL)` Format a value for display, using either sprintf(), format() or a custom formatting function.

`addStyle(styleName, declarations)` Define a new TableStyle and add it to the TableStyles collection.

`createInlineStyle(baseStyleName, declarations)` Create a TableStyle object that can be used to style individual cells in the table.

`setStyling(rFrom=NULL, cFrom=NULL, rTo=NULL, cTo=NULL, cells=NULL, baseStyleName=NULL, style=NULL, decl`
Set the style settings across a range of cells.

`resetCells()` Clear the cells of the table.

`getCells(specifyCellsAsList=TRUE, rowNumbers=NULL, columnNumbers=NULL, cellCoordinates=NULL)`
Retrieve cells by a combination of row and/or column numbers.

`findCells(rowNumbers=NULL, columnNumbers=NULL, minValue=NULL, maxValue=NULL, exactValues=NULL, include`
Find cells in the body of the table matching the specified criteria.

`print(asCharacter=FALSE)` Either print the table to the console or retrieve it as a character value.

`asMatrix(firstRowAsColumnNames=FALSE, firstColumnAsRowNames=FALSE, rawValue=FALSE)`
Gets the table as a matrix, with or without headings.

`asDataFrame(firstRowAsColumnNames=FALSE, firstColumnAsRowNames=FALSE, rawValue=FALSE, stringsAsFactor`
Gets the table as a data frame, with or without headings.

`getCss(styleNamePrefix)` Get the CSS declarations for the entire table.

`getHtml(styleNamePrefix)` Get the HTML representation of the table, specifying the CSS style name prefix to use.

`saveHtml(filePath, fullPageHTML=TRUE, styleNamePrefix)` Save the HTML representation of the table to a file.

`renderTable(width, height, styleNamePrefix)` Render the table as a htmlwidget.

`writeToExcelWorksheet(wb=NULL, wsName=NULL, topRowNumber=NULL, leftMostColumnNumber=NULL, mapStylesFr`
Output the table into the specified workbook and worksheet at the specified row-column location.

`asList()` Get a list representation of the table.

`asJSON()` Get a JSON representation of the table.

`viewJSON()` View the JSON representation of the table.

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

http://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.

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.

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,
  fontName
)
```

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).
`fontName`	The name of the font to use, or a comma separated list (for font-fall-back).

Value

A TableStyles object.

Examples

```
orangeColors <- list(
  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))
tbl$theme <- getSimpleColoredTblTheme(parentTable=tbl,
    colors=orangeColors, fontName="Garamond, arial")
tbl$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 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.

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.

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,
  columnFormats = 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.
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.
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(head(bhmsummary))
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,
  columnFormats = 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.
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.
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(head(bhmsummary))
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 *A 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.

Value

Object of R6Class with properties and methods that define a single table cell

Fields

`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).

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

`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

Documentation For more complete explanations and examples please see the extensive vignettes supplied with this package.

`new(...)` Create a new table cell, specifying the field values documented above.

`getCopy())` Get a copy of this cell.

`asList())` Get a list representation of this cell

`asJSON()` Get a JSON representation of this cell

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 *A 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.

Value

Object of R6Class with properties and methods that help manage cell ranges.

Fields

`parentTable` Owning table.

`ranges` A list of cell ranges.

Methods

Documentation For more complete explanations and examples please see the extensive vignettes supplied with this package.

`new(...)` Create a new object to manage cell ranges.

`addRange(rFrom, cFrom, rSpan=NULL, cSpan=NULL, rTo=NULL, cTo=NULL)` Add a cell range to the list of cell ranges.

`findIntersectingRange(rFrom, cFrom, rSpan=NULL, cSpan=NULL, rTo=NULL, cTo=NULL)` Find a cell range that intersects with the specified cell range.

`deleteRange(r, c)` Delete the cell range covering the specified cell.

`asList()` Get a list representation of this style.

`asJSON()` Get a JSON representation of this style.

Examples

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

TableCells *A class that contains the cells from a table.*

Description

The TableCells class contains all of the TableCell objects that comprise a table.

Format

R6Class object.

Value

Object of R6Class with properties and methods relating to the cells of a table.

Fields

`parentTable` Owning table.

`rows` The rows of cells in the table.

Methods

Documentation For more complete explanations and examples please see the extensive vignettes supplied with this package.

`new(...)` Create a new set of table cells, specifying the field values documented above.

`reset())` Clears and removes all of the cells.

`getCell(r, c))` Get the TableCell at the specified row and column coordinates in the table.

`getValue(r, c))` Get the value at the specified row and column coordinates in the table.

`getRowValues(rowNumber=NULL, columnNumbers=NULL, formattedValue=FALSE, asList=FALSE, rebase=TRUE)` Get a vector or list of the values in a row.

`getColumnValues(columnNumber=NULL, rowNumbers=NULL, formattedValue=FALSE, asList=FALSE, rebase=TRUE)` Get a vector or list of the values in a column.

`setCell(r, c, cellType="cell", rawValue=NULL, formattedValue=NULL, visible=TRUE, baseStyleName=NULL, st` Set the details of a cell in the table.

`setBlankCell(r, c, cellType="cell", visible=TRUE, baseStyleName=NULL, styleDeclarations=NULL, rowSpan=` Set a cell to be empty in the table.

`deleteCell(r, c)` Remove a cell from the table (replacing it with a blank one).

`setValue(r, c, rawValue=NULL, formattedValue=NULL)` Set the value of a cell.

`setRow(rowNumber=NULL, startAtColumnNumber=1, cellTypes=NULL, rawValues=NULL, formattedValues=NULL, fo` Set multiple cells across a row at once.

`setColumn(columnNumber=NULL, startAtRowNumber=1, cellTypes=NULL, rawValues=NULL, formattedValues=NULL` Set multiple cells down a column at once.

`extendCells(rowCount=NULL, columnCount=NULL)` .

`moveCell(r, c, cell))` Move the cell to the specified row and column coordinates in the table.

`insertRow(rowNumber, insertBlankCells=TRUE, headerCells=1, totalCells=0)` Insert a new row (moving the rows underneath down), where headerCells and totalCells control default styling.

`deleteRow(rowNumber=NULL)` Delete a row (moving the rows underneath up.

`insertColumn(columnNumber, insertBlankCells=TRUE, headerCells=1, totalCells=0)` Insert a new column (moving other columns rightwards, where headerCells and totalCells control default styling.

`deleteColumn(columnNumber=NULL)` Delete a column (moving other columns leftwards.

`getCells(specifyCellsAsList=FALSE, rowNumbers=NULL, columnNumbers=NULL, cellCoordinates=NULL)` Retrieve cells by a combination of row and/or column numbers.

`findCells(rowNumbers=NULL, columnNumbers=NULL, minValue=NULL, maxValue=NULL, exactValues=NULL, include` Find cells matching the specified criteria.

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

`asList())` Get a list representation of the table cells.

`asJSON()` Get a JSON representation of the table cells.

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 *A class that renders a table in HTML.*

Description

The TableHtmlRenderer class creates a HTML representation of a table.

Format

`R6Class` object.

Value

Object of `R6Class` with properties and methods that render to HTML.

Fields

`parentTable` Owning table.

Methods

Documentation For more complete explanations and examples please see the extensive vignettes supplied with this package.

`new(...)` Create a new table renderer, specifying the field value documented above.

`getTableHtml(styleNamePrefix=NULL)` Get a HTML representation of the table.

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 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 *A 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.

Format

R6Class object.

Value

Object of R6Class with properties and methods that render to Excel via the openxlsx package

Fields

parentTable Owning table.

Methods

Documentation For more complete explanations and examples please see the extensive vignettes supplied with this package.

new(...) Create a new table renderer, specifying the field value documented above.

writeToCell(wb=NULL, wsName=NULL, rowNumber=NULL, columnNumber=NULL, value=NULL, applyStyles=TRUE, bas
Writes a value to a cell and applies styling as needed.

writeToWorksheet(wb=NULL, wsName=NULL, topRowNumber=NULL, leftMostColumnNumber=NULL, outputValuesAs="
Output the table into the specified workbook and worksheet at the specified row-column location.

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.
```

`TableOpenXlsxStyle` *A 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.

Value

Object of `R6Class` with properties and methods that help define styles.

Fields

`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.

`border` 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.

Methods

Documentation For more complete explanations and examples please see the extensive vignettes supplied with this package.

`new(...)` Create a new Excel style, specifying the field values documented above.

`isBasicStyleNameMatch(baseStyleName=NULL)` Find a matching base style by name.

`isFullStyleDetailMatch = function(baseStyleName=NULL, isBaseStyle=NULL, fontName=NULL, fontSize=NULL,` Find a matching style matching on all the attributes of the style.

`createOpenXslxStyle()` Create the openxlsx style.

`asList()` Get a list representation of this style.

`asJSON()` Get a JSON representation of this style.

`asString()` Get a text representation of this style.

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.
```

`TableOpenXlsxStyles` *A 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.

Value

Object of R6Class with properties and methods that define styles/a theme for a table.

Fields

`parentTable` Owning table.

Methods

Documentation For more complete explanations and examples please see the extensive vignettes supplied with this package.

`new(...)` Create a new set of styles, specifying the field values documented above.

`findNamedStyle(baseStyleName)` Find an existing openxlsx style matching the name of a base style.

`findOrAddStyle(action="findOrAdd", baseStyleName=NULL, isBaseStyle=NULL, style=NULL, mapFromCss=TRUE)` Find an existing openxlsx style and/or add a new openxlsx style matching a base style and/or TableStyle object.

`addNamedStyles(mapFromCss=TRUE)` Populate the OpenXlsx styles based on the styles defined in the table.

`asList()` Get a list representation of the styles.

`asJSON()` Get a JSON representation of the styles.

`asString()` Get a text representation of the styles.

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 *A 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.

Value

Object of R6Class with properties and methods that help define styles.

Fields

`parentTable` Owning table.

`styleName` Style unique name.

`declarations` CSS style declarations.

Methods

Documentation For more complete explanations and examples please see the extensive vignettes supplied with this package.

`new(...)` Create a new style declaration, specifying the field values documented above.

`setPropertyValue(property, value)` Set a single style property.

`setPropertyValues(declarations)` Set multiple style properties, where declarations is a list similar to the code example below.

`getPropertyValue(property)` Get the style declarations for a single property.

`asCSSRule(selector)` Get this style definition in the form of a CSS rule.

`asNamedCSSStyle(styleNamePrefix)` Get this style definition in the form of a named CSS style.

`getCopy(newStyleName)` Get a copy of this style.

`asList()` Get a list representation of this style.

`asJSON()` Get a JSON representation of this style.

Examples

```
# TableStyle objects are normally created indirectly via one of the helper
# methods.
# For an example, see the TableStyles class.
```

TableStyles *A class that defines a collection of styles.*

Description

The TableStyles class defines all of the base styles needed to style/theme a table. It also defines the names of the styles that are used for styling the different parts of the table.

Format

R6Class object.

Value

Object of R6Class with properties and methods that define styles/a theme for a table.

Fields

`parentTable` Owning table.

`themeName` The name of the theme.

`allowExternalStyles` Enables 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

Documentation For more complete explanations and examples please see the extensive vignettes supplied with this package.

`new(...)` Create a new set of styles, specifying the field values documented above.

`isExistingStyle(styleName)` Check whether the specified style exists.

`getStyle(styleName)` Get the specified style.

`addStyle(styleName, declarations)` Add a new style to the collection of styles.

`copyStyle(styleName, newStyleName)` Create a copy of a style with the specified name.

`asCSSRule(styleName, selector)` Get a style definition in the form of a CSS rule.

`asNamedCSSStyle(styleName, styleNamePrefix)` Get a style definition in the form of a named CSS style.

`asList()` Get a list representation of the styles.

`asJSON()` Get a JSON representation of the styles.

`asString()` Get a text representation of the styles.

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

http://www.recenttraintimes.co.uk/

Index

∗Topic **datasets**
bhmsummary, 7
trainstations, 34

BasicTable, 3
basictabler, 5
basictablerOutput, 6
basictablerSample, 6
bhmsummary, 7

checkArgument, 8
cleanCssValue, 9
containsText, 10

getBlankTblTheme, 10
getCompactTblTheme, 11
getDefaultTblTheme, 11
getLargePlainTblTheme, 12
getNextPosition, 12
getSimpleColoredTblTheme, 13
getTblTheme, 14
getXlBorderFromCssBorder, 14
getXlBorderStyleFromCssBorder, 15

isNumericValue, 15
isTextValue, 16

oneToNULL, 16

parseColor, 17
parseCssBorder, 17
parseCssSizeToPt, 18
parseCssSizeToPx, 18
parseCssString, 19
parseXlBorder, 19

qhtbl, 20
qtbl, 21

R6Class, *3*, *23–25*, *27–29*, *31–33*
renderBasictabler, 23

TableCell, 23
TableCellRanges, 24
TableCells, 25
TableHtmlRenderer, 27
TableOpenXlsxRenderer, 28
TableOpenXlsxStyle, 29
TableOpenXlsxStyles, 30
TableStyle, 32
TableStyles, 33
trainstations, 34