Package 'tableHTML'

April 17, 2023

Type Package

Title A Tool to Create HTML Tables

Version 2.1.2

URL https://github.com/LyzandeR/tableHTML

BugReports https://github.com/LyzandeR/tableHTML/issues

Depends R (>= 3.2.0)

Imports htmltools, magrittr, shiny, jpeg, methods, png, webshot

Description A tool to create and style HTML tables with CSS. These can be exported and used in any application that accepts HTML (e.g. 'shiny', 'rmarkdown', 'PowerPoint'). It also provides functions to create CSS files (which also work with shiny).

License MIT + file LICENSE

RoxygenNote 7.2.2

Suggests testthat, knitr, rmarkdown, RColorBrewer

VignetteBuilder knitr

NeedsCompilation no

Author Theo Boutaris [aut, cre, cph],
Clemens Zauchner [aut],
Dana Jomar [aut]

Maintainer Theo Boutaris <teoboot2007@hotmail.com>

Repository CRAN

Date/Publication 2023-04-17 18:30:02 UTC

R topics documented:

`add_css_caption` *Add css to tableHTML's caption*

Description

`add_css_caption` will add css to a tableHTML's caption

Usage

```
add_css_caption(tableHTML, css)
```

Arguments

`tableHTML`	A tableHTML object created by the tableHTML function.
`css`	A list of two elements with the corresponding css. The first element of the list should be an atomic vector with the style definitions (e.g. background-color). The second element will be an atomic vector with the same length as the first element, which will contain the style definitions' values (e.g. red). Check the examples for more information.

Details

`add_css_caption` will add css to a tableHTML's caption.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars, caption = 'This is a caption') %>%
  add_css_caption(css = list(c('color', 'font-size'), c('blue', '50px')))

tableHTML(mtcars, caption = 'This is a caption') %>%
  add_css_caption(css = list(c('color', 'font-size'), c('blue', '50px'))) %>%
  add_css_caption(css = list('background-color', 'green'))
```

---

add_css_column *Add css to tableHTML's columns*

Description

add_css_column will add css to a tableHTML's columns

Usage

```
add_css_column(tableHTML, css, columns)
```

Arguments

tableHTML	A tableHTML object created by the tableHTML function.
css	A list of two elements with the corresponding css. The first element of the list should be an atomic vector with the style definitions (e.g. background-color). The second element will be an atomic vector with the same length as the first element, which will contain the style definitions' values (e.g. red). Check the examples for more information.
columns	A character atomic vector with the names of the columns or a numeric atomic vector with the positions of the columns where the style definitions will be applied on. At least one column must be provided. If the rownames are included the column name is "tableHTML_rownames" and the position is 0. If row_groups are are included the column name is "tableHTML_row_groups" and the position is -1.

Details

add_css_column will add css to a tableHTML's columns. add_css_column will only add css to the columns without the headers or second headers (i.e. it only affects the td tag internally and not the th tag). If you want to add css to the headers or second headers please use add_css_header or add_css_second_header.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars) %>%
 add_css_column(css = list(c('background-color', 'border'), c('lightgray', '3px solid green')),
                columns = 'mpg')

tableHTML(mtcars) %>%
 add_css_column(css = list(c('background-color', 'border'), c('lightgray', '3px solid green')),
                columns = c('mpg', 'disp', 'rownames'))

tableHTML(mtcars,
          rownames = FALSE,
          widths = c(120, rep(50, 11)),
          row_groups = list(c(10, 10, 12), c('Group 1', 'Group 2', 'Group 3'))) %>%
  add_css_column(css = list('background-color', 'lightgray'), columns = 'row_groups') %>%
  add_css_header(css = list('background-color', 'lightgray'), headers = 1)

tableHTML(mtcars,
          rownames = TRUE,
          widths = c(140, rep(50, 12)),
          row_groups = list(c(10, 10, 12), c('Group 1', 'Group 2', 'Group 3')),
          second_headers = list(c(3, 4), c('col1', 'col2'))) %>%
  add_css_column(list('background-color', 'green'), -1) %>%
  add_css_column(list('background-color', 'red'), c(0, 1))
```

---

add_css_conditional_column

Add conditional css to tableHTML's columns

Description

add_css_conditional_column will add conditional css to a tableHTML's columns

Usage

```
add_css_conditional_column(
  tableHTML,
  columns,
 conditional = c("color_rank", "==", "!=", "min", "max", "top_n", "bottom_n", ">", ">=",
    "<", "<=", "between", "contains", "logical"),
  n = NULL,
  value = NULL,
  between = NULL,
  css = NULL,
  color_rank_theme = c("Custom", "RAG", "Spectral", "Rainbow", "White-Green",
```

```
    "White-Red", "White-Blue"),
  color_rank_css = NULL,
  decreasing = FALSE,
  same_scale = TRUE,
  logical_conditions = NULL,
  levels = NULL
)
```

Arguments

tableHTML	A tableHTML object created by the tableHTML function.
columns	A character atomic vector with the names of the columns or a numeric atomic vector with the positions of the columns where the style definitions will be applied on. At least one column must be provided. If the rownames are included the column name is "tableHTML_rownames" and the position is 0. If row_groups are are included the column name is "tableHTML_row_groups" and the position is -1.
conditional	Choose a conditional that should be used to apply css to rows in columns. '==' and '!=' evaluate equality and inequality resp. '<', '<=', '>', and '>=' evaluate the respective operators with the values of columns on the left. 'between' is SQL like, i.e. inclusive. 'top_n' highlights the n highest values columns, 'bottom_n' hightlights the lowest n values. 'max' and 'min' are equivalent of top_1 and bottom_1. 'contains' uses `grepl()` to see if values of a column contain a pattern specified in `value`. 'color-rank' applies one of the `color_rank_theme`. 'logical' allows the user to provide a list of logical vectors to identify where to apply the css. This option is convenient when the condition is complex, for example if it relies on other columns in the table.
n	the number of rows to highlight in 'top_n' and 'bottom_n'. If no value for n is provided, 1 is assumed with a warning.
value	the comparison value for "==", "!=", ">", ">=", "<", "<=", and "contains". value is the right hand side of the equation or the pattern in case of "contains".
between	a numeric vector of length 2 that defines a range, where `between[1]` is the lower bound and `between[2]` is the upper bound of the range. between is inclusive.
css	A list of two elements with the corresponding css. The first element of the list should be an atomic vector with the style definitions (e.g. background-color). The second element will be an atomic vector with the same length as the first element, which will contain the style definitions' values (e.g. red). Check the examples for more information.
color_rank_theme	You can either pick one of the provided themes (RAG, White-Red, White-Green, White-Blue, Spectral, or Rainbow) or create your own by choosing 'Custom' and providing a custom css list in `color_rank_css`.
color_rank_css	An optional named list with css to be applied if a custom styling should be used. The names correspond to a header of the tableHTML, 'rownames', or 'row_groups'. The elements of this css named list are themselves lists of an atomic vector with style definitions (e.g. background-color). and a list of atomic

vecors that contains the style definitions' values with the same length as the number of rows for each style definition. You can use `make_css_color_rank_theme` to obtain this list.

`decreasing` logical. Should the sort order be increasing or decreasing? For the `"radix"` method, this can be a vector of length equal to the number of arguments in `...`. For the other methods, it must be length one.

`same_scale` Logical. This flag indicates whether the condition should be applied to columns individually or in conjunction. If TRUE, the condition will be evaluated on all values of all `columns`. If FALSE, the condition will be evaluated per column.

`logical_conditions`
A list of logical vectors indicating where the condition holds in each column provided in the `columns` parameter. Should be provided when `conditional` is 'logical'. The length of the list should have the same length as `columns`, and the length of each vector in the list should equal the number of rows in the table. If one logical vector is given it wil be applied on all given columns.

`levels` Deprecated. Please change the factor levels in the input data of `tableHTML`.

Details

`add_css_conditional_column` will add conditional css to a tableHTML's columns. `add_css_conditional_column` will only add css to the columns without the headers or second headers (i.e. it only affects the td tag internally and not the th tag). If you want to add css to the headers or second headers please use `add_css_header` or `add_css_second_header`. If you want to apply the same css for all rows in a column, please use `add_css_column`.

Value

A tableHTML object.

Examples

```
qu_25_75 <- quantile(mtcars$disp, c(0.25, 0.75))

tableHTML(mtcars) %>%
  add_css_conditional_column(conditional = "<",
                             value = qu_25_75[1],
                             css = list('background-color', "green"),
                             columns = c("disp")) %>%
  add_css_conditional_column(conditional = "between",
                             between = qu_25_75,
                             css = list('background-color', "orange"),
                             columns = c("disp")) %>%
  add_css_conditional_column(conditional = ">",
                             value = qu_25_75[2],
                             css = list('background-color', "red"),
                             columns = c("disp"))

tableHTML(mtcars) %>%
  add_theme('rshiny-blue') %>%
```

```
  add_css_header(css = list(c("background-color", "color"),
                 c("darkgray", "white")),
                 headers = 1:12) %>%
  add_css_conditional_column(conditional = "min",
                             css = list('background-color', "#99CCA0"),
                             columns = c("wt")) %>%
  add_css_conditional_column(conditional = "max",
                             value = qu_25_75[1],
                             css = list('background-color', "#EA9393"),
                             columns = c("disp")) %>%
  add_css_conditional_column(conditional = "contains",
                             value = "Toyota",
                              css = list(c('background-color', "color"),
                                         c("lightgrey", "darkred")),
                              columns = c("rownames"))  %>%
  add_css_conditional_column(conditional = "contains",
                             value = "Mazda",
                             css = list(c('background-color', "color"),
                                        c("steelblue", "lightgray")),
                             columns = c("rownames")) %>%
  add_css_conditional_column(conditional = "color_rank",
                             color_rank_theme = "White-Blue",
                             columns = 11)

tableHTML(mtcars) %>%
  add_theme('scientific') %>%
  add_css_conditional_column(conditional = "color_rank",
                             color_rank_theme =  "RAG",
                             columns = 1) %>%
  add_css_conditional_column(conditional = "color_rank",
                             color_rank_theme = "Rainbow",
                             columns = 5,
                             decreasing = TRUE)

css <- make_css_color_rank_theme(list(qsec = mtcars$qsec),
                                 colors = c('#E41A1C', '#377EB8', '#4DAF4A',
                                            '#984EA3', '#FF7F00', '#FFFF33',
                                            '#A65628', '#F781BF', '#999999'))

tableHTML(mtcars) %>%
  add_css_conditional_column(conditional = "color_rank",
                             color_rank_theme =  "Custom",
                             color_rank_css = css,
                             columns = 7,
                             decreasing = FALSE,
                             same_scale = FALSE)

tableHTML(mtcars) %>%
  add_css_conditional_column(conditional = "color_rank",
                             color_rank_theme = "RAG",
                             columns = c(1, 5)) %>%
  add_css_conditional_column(conditional = "color_rank",
                             color_rank_theme = "White-Blue",
```

```
                              columns = c(8, 11),
                              same_scale = TRUE) %>%
  add_css_conditional_column(conditional = "color_rank",
                              color_rank_theme = "White-Red",
                              columns = c(9, 10),
                              same_scale = FALSE)

# test the condition on a column and apply the css on another
iris %>%
  tableHTML(rownames = FALSE,
            widths = rep(100, ncol(iris))) %>%
  add_css_conditional_column(
   conditional = 'logical',
   columns = c('Sepal.Length', 'Petal.Length'),
   css = list(c('background-color'), c('lightblue')),
   logical_conditions = list(iris$Sepal.Width==3,
                              iris$Petal.Width==0.3))
# apply the css on a full row
iris %>%
  tableHTML(rownames = FALSE,
            widths = rep(100, ncol(iris))) %>%
  add_css_conditional_column(conditional = 'logical',
                              columns = 1:ncol(iris),
                              css = list(c('background-color'), c('lightblue')),
                              logical_conditions = list(iris$Sepal.Width==3))
```

`add_css_footer` *Add css to tableHTML's footer*

Description

`add_css_footer` will add css to a tableHTML's footer

Usage

```
add_css_footer(tableHTML, css)
```

Arguments

`tableHTML`	A tableHTML object created by the tableHTML function.
`css`	A list of two elements with the corresponding css. The first element of the list should be an atomic vector with the style definitions (e.g. background-color). The second element will be an atomic vector with the same length as the first element, which will contain the style definitions' values (e.g. red). Check the examples for more information.

Details

`add_css_footer` will add css to a tableHTML's footer.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars, footer = 'This is a footer') %>%
  add_css_footer(css = list(c('color', 'font-size'), c('blue', '50px')))

tableHTML(mtcars, footer = 'This is a footer') %>%
  add_css_footer(css = list(c('color', 'font-size'), c('blue', '50px'))) %>%
  add_css_footer(css = list('background-color', 'green'))
```

---

add_css_header *Add css to tableHTML's headers*

Description

add_css_header will add css to a tableHTML's headers

Usage

```
add_css_header(tableHTML, css, headers)
```

Arguments

tableHTML	A tableHTML object created by the tableHTML function.
css	A list of two elements with the corresponding css. The first element of the list should be an atomic vector with the style definitions (e.g. background-color). The second element will be an atomic vector with the same length as the first element, which will contain the style definitions' values (e.g. red). Check the examples for more information.
headers	A numeric atomic vector with the indices of the headers where the style definitions will be applied on. At least one header index must be provided.

Details

add_css_header will add css to a tableHTML's headers.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars) %>%
 add_css_header(css = list(c('background-color', 'border'), c('lightgray', '3px solid green')),
                headers = 2)

tableHTML(mtcars) %>%
 add_css_header(css = list(c('background-color', 'border'), c('lightgray', '3px solid green')),
                headers = c(1, 4))
```

---

add_css_row *Add css to tableHTML's rows*

Description

add_css_row will add css to a tableHTML's rows

Usage

```
add_css_row(tableHTML, css, rows = NULL)
```

Arguments

tableHTML	A tableHTML object created by the tableHTML function.
css	A list of two elements with the corresponding css. The first element of the list should be an atomic vector with the style definitions (e.g. background-color). The second element will be an atomic vector with the same length as the first element, which will contain the style definitions' values (e.g. red). Check the examples for more information.
rows	A numeric atomic vector with the indices of the rows on which the style definitions will be applied. headers and second_headers are included in the rows.Default is NULL which means that it will be applied to all rows.

Details

add_css_row will add css to a tableHTML's rows. The only thing you need to be cautious about is the rows argument. headers and second_headers are still considered rows. add_css_row affects the tr tag of the HTML code internally.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars) %>%
 add_css_row(css = list(c('background-color', 'border'), c('lightgray', '3px solid green')))

tableHTML(mtcars) %>%
 add_css_row(css = list(c('background-color', 'border'), c('lightgray', '3px solid green')),
             rows = 1:33)
```

`add_css_rows_in_column` *Add css to tableHTML's columns' rows.*

Description

`add_css_rows_in_column` will add css to a tableHTML's individual rows within a column

Usage

```
add_css_rows_in_column(tableHTML, css, column)
```

Arguments

`tableHTML`	A tableHTML object created by the tableHTML function.
`css`	A list of two elements. The first element of the list should be an atomic vector of length 1 with the style definition (e.g. background-color). The second element will be an atomic vector with the same length as the column, which will contain the style definitions' values (e.g. red). Check the examples for more information.
`column`	A character atomic vector of length 1, with the name of the column or a numeric atomic vector with the positions of the columns where the style definitions will be applied on. Only one column must be provided. If the rownames are included the column name is "tableHTML_rownames" and the position is 0. If row_groups are included the column name is "tableHTML_row_groups" and the position is -1.

Details

`add_css_rows_in_column` will add css to a tableHTML's individual rows within a column. Only one css style definition can be used, and multiple values (same length as the column) will be applied to the rows within the column. As an example a list of different colours can be provided for all the rows within a column.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars) %>%
  add_css_rows_in_column(css = list('background-color',
                                    rep(c('red', 'green'), each = 16)),
                         column = 'mpg')

tableHTML(mtcars) %>%
  add_css_column(css = list('border', '3px solid blue'),
                 columns = c('mpg', 'disp', 'rownames')) %>%
  add_css_rows_in_column(css = list(c('background-color'),
                                    rep(c('red', 'green'), each = 16)),
                         column = 'mpg')


tableHTML(mtcars) %>%
  add_css_rows_in_column(css = list(c('background-color'),
                                    rep(c('red', 'green'), each = 16)),
                         column = 'mpg') %>%
  add_css_column(css = list('border', '3px solid blue'),
                 columns = c('mpg', 'disp', 'rownames'))
```

`add_css_second_header` *Add css to tableHTML's second headers*

Description

`add_css_second_header` will add css to a tableHTML's second headers

Usage

```
add_css_second_header(tableHTML, css, second_headers)
```

Arguments

`tableHTML`	A tableHTML object created by the tableHTML function.
`css`	A list of two elements with the corresponding css. The first element of the list should be an atomic vector with the style definitions (e.g. background-color). The second element will be an atomic vector with the same length as the first element, which will contain the style definitions' values (e.g. red). Check the examples for more information.
`second_headers`	A numeric atomic vector with the indices of the second headers where the style definitions will be applied on. At least one second header index must be provided.

Details

`add_css_second_header` will add css to a tableHTML's second headers.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars, second_headers = list(c(3, 4, 5), c('col1', 'col2', 'col3'))) %>%
  add_css_second_header(css = list(c('background-color', 'border'),
                                   c('lightgray', '3px solid green')),
                        second_headers = c(1, 3))
```

---

add_css_table *Add css to the whole tableHTML*

Description

`add_css_table` will add css to the whole HTML table

Usage

```
add_css_table(tableHTML, css)
```

Arguments

`tableHTML`	A tableHTML object created by the tableHTML function.
`css`	A list of two elements with the corresponding css. The first element of the list should be an atomic vector with the style definitions (e.g. background-color). The second element will be an atomic vector with the same length as the first element, which will contain the style definitions' values (e.g. red). Check the examples for more information.

Details

`add_css_table` will add css to the whole HTML table.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars) %>%
  add_css_table(css = list('background-color', 'lightgray'))

tableHTML(mtcars) %>%
  add_css_table(css = list('background-color', 'lightgray')) %>%
  add_css_table(css = list('background-color', 'lightblue'))
```

```
tableHTML(mtcars) %>%
  add_css_table(css = list('background-color', 'lightgray')) %>%
  add_css_table(css = list('text-align', 'center'))
```

---

add_css_tbody *Add css to the tbody tag*

Description

add_css_tbody will add css to the tbody tag i.e. to all table apart from the headers and second headers.

Usage

```
add_css_tbody(tableHTML, css)
```

Arguments

tableHTML	A tableHTML object created by the tableHTML function.
css	A list of two elements with the corresponding css. The first element of the list should be an atomic vector with the style definitions (e.g. background-color). The second element will be an atomic vector with the same length as the first element, which will contain the style definitions' values (e.g. red). Check the examples for more information.

Details

add_css_thead will add css to the tbody tag i.e. to all table apart from the headers and second headers.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars) %>%
  add_css_tbody(css = list('background-color', 'lightgray'))

tableHTML(mtcars) %>%
  add_css_tbody(css = list('background-color', 'lightgray')) %>%
  add_css_tbody(css = list('background-color', 'lightblue'))

tableHTML(mtcars) %>%
  add_css_tbody(css = list('background-color', 'lightgray')) %>%
  add_css_tbody(css = list('text-align', 'center'))
```

add_css_thead *Add css to the thead tag*

Description

add_css_thead will add css to the thead tag i.e. to headers and second_headers.

Usage

```
add_css_thead(tableHTML, css)
```

Arguments

tableHTML	A tableHTML object created by the tableHTML function.
css	A list of two elements with the corresponding css. The first element of the list should be an atomic vector with the style definitions (e.g. background-color). The second element will be an atomic vector with the same length as the first element, which will contain the style definitions' values (e.g. red). Check the examples for more information.

Details

add_css_thead will add css to the thead tag i.e. to headers and second_headers.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars) %>%
  add_css_thead(css = list('background-color', 'lightgray'))

tableHTML(mtcars) %>%
  add_css_thead(css = list('background-color', 'lightgray')) %>%
  add_css_thead(css = list('background-color', 'lightblue'))

tableHTML(mtcars) %>%
  add_css_thead(css = list('background-color', 'lightgray')) %>%
  add_css_thead(css = list('text-align', 'center'))
```

add_editable_column *Make columns Editable*

Description

add_editable_column will make the specified columns editable

Usage

```
add_editable_column(tableHTML, columns)
```

Arguments

tableHTML	A tableHTML object created by the tableHTML function.
columns	A character atomic vector with the names of the columns or a numeric atomic vector with the positions of the columns where the style definitions will be applied on. At least one column must be provided. If the rownames are included the column name is "tableHTML_rownames" and the position is 0. If row_groups are are included the column name is "tableHTML_row_groups" and the position is -1.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars) %>%
  add_editable_column(columns = 'mpg')

tableHTML(mtcars,
          rownames = TRUE,
          widths = c(150, 100, rep(50, 11)),
          row_groups = list(c(10, 10, 12), c('Group 1', 'Group 2', 'Group 3'))) %>%
  add_css_column(css = list('background-color', 'lightgray'), columns = 'row_groups') %>%
   add_css_column(css = list('text-align', 'right'), columns = 'row_groups') %>%
   add_css_header(css = list('background-color', 'lightgray'), headers = 1) %>%
   add_editable_column(columns = -1:3)
```

add_theme *Add a theme to the tableHTML*

Description

add_theme will add a theme to tableHTML

Usage

```
add_theme(tableHTML, theme = c("scientific", "rshiny-blue", "colorize"), ...)
```

Arguments

tableHTML	A tableHTML object.
theme	Pick one of the provided themes. These can still be modified by extra css. Choices are: scientific, rshiny-blue, colorize. Column widths are not provided when you select a theme. Please use the width argument for column widths.
...	Additional parameters to pass to the theme. Currently "colorize" is the only theme that takes additional parameters. For more details on those parameters see add_theme_colorize.

Details

add_theme will add a theme to tableHTML.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars,
          rownames = FALSE,
          widths = c(140, rep(50, 11)),
          row_groups = list(c(10, 10, 12), c('Group 1', 'Group 2', 'Group 3')),
          second_headers = list(c(3, 4), c('col1', 'col2'))) %>%
  add_theme('scientific')

tableHTML(mtcars, widths = c(140, rep(50, 11))) %>%
  add_theme ('rshiny-blue')

mtcars %>%
  tableHTML(widths = c(150, rep(50, 11)),
            rownames = TRUE) %>%
  add_theme('colorize')

generate_df <- function(){
  df <- data.frame(Month = month.abb,
```

```
                  x1 = sample(1:100, 12),
                  x2 = sample(1:100, 12),
                  x3 = sample(1:100, 12),
                  stringsAsFactors = FALSE)
  df[nrow(df) + 1, ] <- c('Total', sum(df$x1), sum(df$x2), sum(df$x3))
  return(df)
}
df_1 <- generate_df()
df_2 <- generate_df()

rbind(df_1, df_2) %>%
  tableHTML(widths = rep(50, 4), rownames = FALSE) %>%
  add_theme('colorize', total_rows = c(13, 26),
            color = c('steelblue', 'green3'), id_column = TRUE)
```

`add_theme_colorize` *Add a theme to a tableHTML with a total row.*

Description

`add_theme_colorize` will add an Excel-like theme to tableHTML and hightlights one or more total-rows.

Usage

```
add_theme_colorize(
  tableHTML,
  color = "steelblue",
  total_rows = NULL,
  id_column = FALSE
)
```

Arguments

`tableHTML`	A tableHTML object.
`color`	A character vector to specify the desired color. It can contain at most two colors. Accepts color names (as listed by `colors()`), as well as hexadecimal representation of the form "#rrggbb". If two colors are chosen, the first color will be the dominant one, and row coloring will alternate between the first and second color.
`total_rows`	A numeric atomic vector with the indices of the total/subtotal rows. Default is `NULL` which means no row will be highlighted.
`id_column`	A boolean, if set to `TRUE` the first column will be highlighted as an ID column. Default is `FALSE`.

Details

`add_theme_colorize` will add an Excel-like theme to tableHTML. Column widths are not provided with the theme. Please use the width argument for column widths.

Value

A tableHTML object.

Examples

```
# no total rows
mtcars %>%
 tableHTML(widths = c(140, rep(50, 11))) %>%
 add_theme_colorize()

# one total row
x1 <- sample(1:100, 12)
x2 <- sample(1:100, 12)
x3 <- sample(1:100, 12)

df <- data.frame(Month = month.abb, x1, x2, x3,
                 stringsAsFactors = FALSE)

df[nrow(df) + 1, ] <- c('Total', sum(x1), sum(x2), sum(x3))

df %>%
  tableHTML(widths = rep(50, 4), rownames = FALSE) %>%
  add_theme_colorize(total_rows = nrow(df),
                     color = 'darkred')


# multiple subtotal rows
df_q <- rbind(
  df[1:3, ],
  c('Sum1', sum(x1[1:3]), sum(x2[1:3]), sum(x3[1:3])),
  df[4:6, ],
  c('Sum2', sum(x1[4:6]), sum(x2[4:6]), sum(x3[4:6])),
  df[7:9, ],
  c('Sum3', sum(x1[7:9]), sum(x2[7:9]), sum(x3[7:9])),
  df[10:12, ],
  c('Sum4', sum(x1[10:12]), sum(x2[10:12]), sum(x3[10:12])))

df_q %>%
  tableHTML(widths = rep(50, 5),
            rownames = FALSE,
            row_groups = list(c(4, 4, 4, 4),
                              c('Q1', 'Q2', 'Q3', 'Q4'))) %>%
  add_theme_colorize(color = '#009999',
                   total_rows = c(4, 8, 12, 16))

# Two colors and an id_column
df_q %>%
  tableHTML(widths = rep(50, 5),
            rownames = FALSE,
            row_groups = list(c(4, 4, 4, 4),
                              c('Q1', 'Q2', 'Q3', 'Q4'))) %>%
  add_theme_colorize(color = c('pink3', 'yellow2'),
```

```
total_rows = c(4, 8, 12, 16), id_column = TRUE)
```

`create_hexlogo` *Generate hexagon logo from a tableHTML object.*

Description

The purpose of the function `create_hexlogo` is to generate the hexagon logo of the `tableHTML` package.

Usage

```
create_hexlogo(
  save = TRUE,
  format = "html",
  file = "tableHTML_hexlogo.html",
  complete_html = FALSE,
  ...
)
```

Arguments

`save`	A boolean when set to TRUE the logo will be saved in the specified format.
`format`	A character string to specify the format of the output, it accepts `'html'`, `'png'`, or `'jpeg'`. Default is `'html'`.
`file`	A character string to specify the name and path to the new file. Should end with `'.html'`, `'.png'`, or `'.jpeg'`, depending on the selected format.
`complete_html`	Either TRUE or FALSE. Defaults to FALSE. If TRUE then the <html> and <body> tags are also added in the file.
`...`	Further parameters to pass to webshot.

Details

The function `create_hexlogo` will generate the hexagon logo of the `tableHTML` package.

Value

The hexagon logo of the tableHTML package as a tableHTML object.

The output will be shown in the Viewer pane, and can be saved either as an image or as an HTML.

Examples

```
## Not run:
create_hexlogo(save = FALSE)

create_hexlogo(format = 'jpeg',
            file = '~/exported_hexlogo.jpeg')

## End(Not run)
```

---

create_logo *Generate package's logo*

Description

create_logo will create and generate the package's logo.

Usage

```
create_logo(
  save = TRUE,
  format = "html",
  file = "tableHTML_logo.html",
  complete_html = FALSE,
  ...
)
```

Arguments

save	A boolean when set to TRUE the logo will be saved in the specified format.
format	A character string to specify the format of the output, it accepts 'html', 'png', or 'jpeg'. Default is 'html'.
file	A character string to specify the name and path to the new file. Should end with '.html', '.png', or '.jpeg', depending on the selected format.
complete_html	Either TRUE or FALSE. Defaults to FALSE. If TRUE then the <html> and <body> tags are also added in the file.
...	Further parameters to pass to webshot.

Details

create_logo will create and generate the package's logo.

Value

The logo of the tableHTML package as a tableHTML object. The output will be shown in the Viewer pane, and can be saved either as an image or as an HTML.

Examples

```
## Not run:
create_logo(save = FALSE)

create_logo(format = 'png', file = '~/exported_logo.png')

## End(Not run)
```

`make_css` *Create a css file or string*

Description

`make_css` will create a css file or string which can also be used in shiny

Usage

```
make_css(..., file = NULL)
```

Arguments

`...`	css style definitions. Each object you provide must be a list of three elements. The first element will be a vector of the selectors to be styled (e.g. table, th, an id or html class). If the first element is a vector of length greater than one then the selectors will be comma separated in the css. The second element will be a vector of the css definitions and the third element will a vector of the values of those definitions.
`file`	Character sting. If a file name is provided then the css code will be printed into that file. If the argument is NULL (default) then a string will be returned.

Details

`make_css` will create a css file or string which can also be used in shiny. If the argument file is provided the css code will be printed out to the file. The file can then be used in shiny with the `includeCSS` function. Alternatively there are two ways to use `make_css` in order to add css to your shiny app. If you have a very small css file or you are just testing your app you can use `tags$style` with `make_css` directly. There is an example in the examples section. Another way (which will make your code cleaner) is to create your css in global.R assign it to a variable and then use that variable with `tags$style`. There is another example on the examples section. Keep in mind that for complete shiny apps it is best practice to use a file and load it with `includeCSS`. This will be faster as well as it won't run the code to create the css file each time.

Value

css definitions.

Examples

```
make_css(list('table', c('text-align', 'font-size'), c('center', '20px')),
         list('th', c('background-color', 'height'), c('lightgreen', '30px')))

make_css(list(c('table', 'td'), c('text-align', 'font-size'), c('center', '20px')),
         list('th', c('background-color', 'height'), c('lightgreen', '30px')))

make_css(list('tr:hover', c('text-align', 'font-size'), c('center', '20px')),
         list('th', c('background-color', 'height'), c('lightgreen', '30px')))
```

---

make_css_color_rank_theme

Get css properties for custom color rank theme

Description

make_css_color_rank_theme will create a list of css properties needed for custom conditional formatting.

Usage

```
make_css_color_rank_theme(
  column_data,
  colors,
  css_property = "background-color",
  decreasing = FALSE,
  same_scale = TRUE
)
```

Arguments

column_data	A named list of vectors of values that are in a tableHTML column which should be mapped to a color palette.
colors	colors to interpolate; must be a valid argument to col2rgb().
css_property	Character. An optional character specifying the css attribute that should be used. Default is 'backgroud-color'
decreasing	logical. Should the sort order be increasing or decreasing? For the "radix" method, this can be a vector of length equal to the number of arguments in .... For the other methods, it must be length one.
same_scale	Logical. This flag indicates whether the condition should be applied to columns individually or in conjunction. If TRUE, the condition will be evaluated on all values of all columns. If FALSE, the condition will be evaluated per column.

Details

make_css_color_rank_theme will add conditional css to a tableHTML's columns. add_conditional_css_column will only add css to the columns without the headers or second headers (i.e. it only affects the td tag internally and not the th tag). If you want to add css to the headers or second headers please use add_css_header or add_css_second_header.

Value

A list of css properties

Examples

```
tableHTML <- tableHTML(mtcars)

css <- make_css_color_rank_theme(list(mpg = mtcars$mpg),
                                 c("orange", "yellow","springgreen","royalblue"))

tableHTML %>% add_css_conditional_column(conditional = "color_rank",
                                         color_rank_theme = "Custom",
                                         color_rank_css = css, column = 1)
```

---

make_hyperlink *Create a hyperlink to display as a link on a tableHTML*

Description

make_hyperlink add the relevant <a> tag on a vector.

Usage

```
make_hyperlink(vec, message = NULL)
```

Arguments

vec	A vector. Typically the column of a data.frame.
message	The hyperlinks name. If it is omitted, the actual link will be used as the name

Details

make_hyperlink The standard way to use this (although it can be used outside a tableHTML) is to convert a column with plain URLs into clickable hyperlinks when you render the HTML table. **Make sure the escape argument of tableHTML is set to FALSE for this to work as expected.**

Value

A character vector which will represent an HTML hyperlink

Examples

```
#make sure the escape argument is set to FALSE for this to work
tableHTML(data.frame(mpg = make_hyperlink(mtcars$mpg)), escape = FALSE)

tableHTML(data.frame(mpg = make_hyperlink(mtcars$mpg, 1:32)), escape = FALSE)
```

---

odd *Get the odd or even numbers from a numeric vector*

Description

Get the odd or even numbers from a numeric vector

Usage

```
odd(vec)

even(vec)
```

Arguments

vec	A numeric atomic vector.

Details

odd will extract the odd numbers from a vector.
even will extract the even numbers from a vector

Value

A numeric atomic vector with the odd / even numbers

Examples

```
odd(1:10)
even(1:10)
```

render_tableHTML *Implementing tableHTML in shiny*

Description

This function is used to implement tableHTML in a shiny app. This function is used in the shiny server.R file. Internally, it just calls renderUI, since tableHTML creates HTML code.

Usage

```
render_tableHTML(expr, ...)
```

Arguments

expr	A tableHTML object.
...	Other arguments passed along to shiny::renderUI().

See Also

shiny::renderUI()

Examples

```
## Not run:

library(shiny)
shinyApp(
 ui = fluidPage(
  fluidRow(
  #leave some spacing
  br(),
  column(width = 1),
  tableHTML_output("mytable"))
 ),
server = function(input, output) {
 output$mytable <- render_tableHTML(
  tableHTML(mtcars)
 )}
)


## End(Not run)
```

replace_html *Replaces a tableHTML string with another*

Description

replace_html replaces a tableHTML string with another

Usage

```
replace_html(tableHTML, pattern, replacement, replace_all = FALSE, ...)
```

Arguments

tableHTML	A tableHTML object created by the tableHTML function.
pattern	A tableHTML string to be replaced. Regex is allowed.
replacement	A replacement for the matched pattern.
replace_all	TRUE or FALSE. If TRUE gsub is used internally and all the pattern occurrances will be replaced. If FALSE sub is used internally and only the first occurance will be replaced. Defaults to FALSE.
...	Additional arguments passed on to sub or gsub.

Details

replace_html replaces a tableHTML string with another. The function calls sub and gsub internally (according to the replace_all argument) to do the replacements but in a safe way in order to preserve the class of the tableHTML object. Also, replace_html has been developed so that it can be used with chaining (using the pipe operator %>%). See the examples to understand exactly how.

Value

A tableHTML object.

See Also

gsub or sub

Examples

```
a <- mtcars %>%
  tableHTML() %>%
  add_css_row(css = list('background-color', 'lightblue'), rows = 1)

a %>%
  replace_html('lightblue', 'green')
```

tableHTML *Create an easily css-ible HTML table*

Description

The purpose of tableHTML is to create easily css-ible HTML tables that are compatible with R shiny. The exported HTML table will contain separate ids or classes for headers, columns, second headers (if any) and the table itself (in case you have multiple tables) in order to create a complex css file very easily. ids and classes are explained in detail in the details section.

Usage

```
tableHTML(
  obj,
  rownames = TRUE,
  class = paste0("table_", sample(1000:9999, 1)),
  widths = NULL,
  headers = NULL,
  second_headers = NULL,
  row_groups = NULL,
  caption = NULL,
  footer = NULL,
  border = 1,
  collapse = c("collapse", "separate", "separate_shiny"),
  spacing = "2px",
  escape = TRUE,
  round = NULL,
  replace_NA = NULL,
  add_data = TRUE,
  theme = NULL
)

## S3 method for class 'tableHTML'
print(x, viewer = TRUE, ...)
```

Arguments

obj	Needs to be a data.frame or a matrix or an arbitrary object that has the data.frame class and can be coersible to a data.frame (e.g data.table, tbl, etc.)
rownames	Can be TRUE or FALSE. Defaults to TRUE. Whether the obj's rownames will be inlcuded.
class	Character string. Specifies the table's class. Convinient if you have multiple tables. Default is table_xxxx (random 4-digit number).
widths	Needs to be a numeric atomic vector with the column widths. Widths are in pixels.

headers	character vector. The headers for the HTML table. If not provided the original data.frame headers will be used.
second_headers	A list of two elements of the same length. The first element will contain the column spans (i.e. a numeric atomic vector) whereas the second element will contain the names (i.e. a character atomic vector). See the examples for more info. Defauls to NULL.
row_groups	A list of two elements of the same length. The first element will contain the row spans (i.e. a numeric atomic vector) whereas the second element will contain the names (i.e. a character atomic vector). See the examples for more info. Defauls to NULL.
caption	Character string. The table's caption.
footer	Character string. The table's footer. This gets added below the table and it should not be confused with tfooter.
border	An integer. Specifies the border of the table. Defaults to 1. 0 removes borders from the table. The higher the number the thicker the table's outside border.
collapse	Whether to collapse the table or not. By default the tables are collapsed. The choices for this argument are 'collapse', 'separate' and 'separate_shiny'. Check the details about which one to use.
spacing	Character string. This is only used if collapse is either separate or separate_shiny and sets the spacing between the table's cells. It defaults to 2px. Can be one or two length values (provided as a string). If two length values are provided the first one sets the horizontal spacing whereas the second sets the vertical spacing. See the examples.
escape	Can be TRUE or FALSE. Defaults to TRUE. Escapes characters < and > because they can close (or open) the table's HTML tags if they exist within the data.frame's text. This means that all < and > characters within the tableHTML will be converted to &#60 and &#62 respectively.
round	An integer specifying the number of decimals of numbers of numeric columns only. Defaults to NULL which means no rounding.
replace_NA	A sting that specifies with what to replace NAs in character or factor columns only. Defaults to NULL which means NAs will be printed.
add_data	TRUE or FALSE. Defaults to TRUE. If set to true, the data.frame or matrix passed in `obj` will be added to the attributes. If set to FALSE, the object will be smaller, but `add_css_conditional_column` would not be applicable.
theme	Argument is Deprecated. Please use the add_theme function instead.
x	A tableHTML object created from the `tableHTML` function.
viewer	TRUE or FALSE. Defaults to TRUE. Whether or not to render the HTML table. If you are working on Rstudio (interactively) the table will be printed or Rstudio's viewer. If you are working on Rgui (interactively) the table will be printed on your default browser. If you set this to FALSE the HTML code will be printed on screen.
...	Optional arguments to print.

Details

tableHTML will create an HTML table with defined ids and classes for rows and columns. In particular:

- **Table:** Will get the class from the class argument in the function.
- **Columns:** Will get an id which will be of the form tableHTML_column_x (where x is the column position). If rownames exist these will get the tableHTML_rownames id. If row groups exist these will get the tableHTML_row_groups id. Check the add_css_column function for examples.
- **Headers:** Will get an id of the form tableHTML_header_x (where x is the header position). For example the first header will have the id tableHTML_header_1, the second header will have tableHTLM_header_2 and so on. If rownames exist these will get the tableHTML_header_0 id.
- **Second_Header:** Will get an id of the form tableHTML_second_header_x (where x is the second header position). For example the first second_header will have the id tableHTML_second_header_1, the second header will have tableHTML_second_header_2 and so on.

Notice that rows do not get a specific id or class.

If you would like to use a non-collapsed table i.e. leave spacing between cells, then you would need to use the collapse argument. Setting it to separate would create a non-collapsed table. However, this choice will not work in shiny. The reason is that shiny uses table {border-collapse: collapse; border-spacing:0;} in its css by default through bootstrap 3. In order to overcome this problem in shiny, collapse needs to be set to separate_shiny instead of separate. By setting collapse to separate_shiny tableHTML uses !important in order to overwrite the standard behaviour of bootstrap 3. !important needs to be used with caution since it overwrites css styles, so unless you are using shiny (or any other place where the above css is automatically loaded) you should be using collapse = 'separate'.

Printing the table will result in rendering it in R studio's viewer with the print.tableHTML method if using Rstudio otherwise it will use the default browser. Use print(tableHTML(obj), viewer = FALSE) or str(tableHTML(obj)) to view the actual html code.

Value

A tableHTML object.

Examples

```
tableHTML(mtcars)
tableHTML(mtcars, rownames = FALSE)
tableHTML(mtcars, class = 'table1')
tableHTML(mtcars, second_headers = list(c(3, 4, 5), c('col1', 'col2', 'col3')))
tableHTML(mtcars,
          widths = c(rep(50, 6), rep(100, 6)),
          second_headers = list(c(3, 4, 5), c('col1', 'col2', 'col3')))
tableHTML(mtcars, caption = 'This is a caption', footer = 'This is a footer')
tableHTML(mtcars,
          row_groups = list(c(10, 10, 12), c('Group 1', 'Group 2', 'Group 3')),
          widths = c(200, rep(50, 5), rep(100, 6)),
          rownames = FALSE)
```

```
tableHTML(mtcars,
          rownames = FALSE,
          widths = c(140, rep(50, 11)),
          row_groups = list(c(10, 10, 12), c('Group 1', 'Group 2', 'Group 3')),
          second_headers = list(c(3, 4, 5), c('col1', 'col2', 'col3')))
tableHTML(mtcars, collapse = 'separate_shiny', spacing = '5px')
tableHTML(mtcars, collapse = 'separate', spacing = '5px 2px')
```

---

tableHTML_output *Implementing tableHTML in shiny*

Description

This function is used to implement tableHTML in a shiny app. It is used in the shiny ui.R file. Internally, it just calls uiOutput, since tableHTML creates HTML code.

Usage

```
tableHTML_output(outputId, ...)
```

Arguments

outputId	input name.
...	Other arguments to passed along to shiny::uiOutput()

See Also

uiOutput()

Examples

```
## Not run:

library(shiny)
shinyApp(
 ui = fluidPage(
  fluidRow(
  #leave some spacing
  br(),
  column(width = 1),
  tableHTML_output("mytable"))
 ),
server = function(input, output) {
 output$mytable <- render_tableHTML(
  tableHTML(mtcars)
 )}
)
```

```
## End(Not run)
```

`tableHTML_to_image` *Convert a tableHTML into an image*

Description

`tableHTML_to_image` converts the tableHTML into an image.

Usage

```
tableHTML_to_image(
  tableHTML,
  file = NULL,
  type = c("png", "jpeg"),
  add = FALSE,
  selector = "table",
  ...
)
```

Arguments

`tableHTML`	A tableHTML object created by the tableHTML function.
`file`	A file to write the image to. If NULL then file is just displayed on screen.
`type`	Either png or jpeg. The type of the image.
`add`	Logical. If TRUE, the plot will be added to the existing plot. If FALSE, the current device will be shut down.
`selector`	One or more CSS selectors specifying a DOM element to set the clipping rectangle to. The screenshot will contain these DOM elements. For a given selector, if it has more than one match, only the first one will be used. This option is not compatible with `cliprect`. When taking screenshots of multiple URLs, this parameter can also be a list with same length as `url` with each element of the list containing a vector of CSS selectors to use for the corresponding URL.
`...`	Parameters passed on to webshot. Check `webshot`.

Details

The main rational behind this function is to make it work well with pdfs / word documents. When using rmarkdown and want to incude a tableHTML in a pdf / word document this is the function you would need to use. Obviously, you don't need this function if you are exporting to an html file.

Specifying a type will determine which function is used to create the image. Either JPEG or PNG. When using JPEG as the type you will need to add a background colour to the table otherwise it will be set to black by JPEG. Both of the built-in themes (rshiny-blue, scientific) work well with JPEG.

When working on rmarkdown and you want to knit as pdf, use this function. Works with microsoft word as well.

To use this function you need phantomjs installed. Please use `webshot::install_phantomjs()` to install if it is not installed already.

Value

An image of the tableHTML.

Examples

```
## Not run:
mtcars %>%
  tableHTML() %>%
  tableHTML_to_image()

## End(Not run)
```

`write_tableHTML` *Writes the HTML code to a file*

Description

`write_tableHTML` will write the HTML code to a file

Usage

```
write_tableHTML(tableHTML, file, complete_html = FALSE)
```

Arguments

`tableHTML`	A tableHTML object created by the tableHTML function.
`file`	A character string. This is the file name. You need to include the extention.
`complete_html`	Either TRUE or FALSE. Defaults to FALSE. If TRUE then the <html> and <body> tags are also added in the file.

Details

`write_tableHTML` will write the HTML code to a file.

Value

The function itself returns nothing but a file is created.

Examples

```
## Not run:
write_tableHTML(tableHTML(mtcars), file = 'myhtmlcode.html')

write_tableHTML(tableHTML(mtcars), file = 'myhtmlcode.html', complete_html = TRUE)

## End(Not run)
```

`%>%` *Pipe css*

Description

Like dplyr and ggvis, tableHTML also uses the pipe function, `%>%` to chain css functions. The pipe function originally came from the magrittr package.

Arguments

`lhs, rhs`	A tableHTML and a function to apply to it

Examples

```
# Instead of
  add_css_row(tableHTML(mtcars),
           css = list(c('background-color', 'border'), c('lightgray', '3px solid green')))
# you can write
  mtcars %>%
    tableHTML() %>%
   add_css_row(css = list(c('background-color', 'border'), c('lightgray', '3px solid green')))
```

Index