Package 'htmlTable'

May 18, 2021

Version 2.2.1

Title Advanced Tables for Markdown/HTML

Maintainer Max Gordon <max@gforge.se>

Description Tables with state-of-the-art layout elements such as row spanners, column spanners, table spanners, zebra striping, and more. While allowing advanced layout, the underlying css-structure is simple in order to maximize compatibility with word processors such as 'MS Word' or 'LibreOffice'. The package also contains a few text formatting functions that help outputting text compatible with HTML/LaTeX.

License GPL (>= 3)

URL https://gforge.se/packages/

BugReports https://github.com/gforge/htmlTable/issues

Biarch yes

Imports stringr, knitr (>= 1.6), magrittr (>= 1.5), methods, checkmate, htmlwidgets, htmltools, rstudioapi (>= 0.6)

Suggests testthat, XML, xml2, Hmisc, reshape, rmarkdown, chron, lubridate, tibble, purrr, tidyselect, glue, rlang, tidyr (>= 0.7.2), dplyr (>= 0.7.4)

Encoding UTF-8

NeedsCompilation no

VignetteBuilder knitr

RoxygenNote 7.1.1

Author Max Gordon [aut, cre], Stephen Gragg [aut], Peter Konings [aut]

Repository CRAN

Date/Publication 2021-05-18 11:10:02 UTC

R topics documented:

`addHtmlTableStyle` *Add/set css and other style options*

Description

This function is a preprocessing step before applying the `htmlTable()` function. You use this to style your tables with HTML cascading style sheet features.

Usage

```
addHtmlTableStyle(
  x,
  align = NULL,
  align.header = NULL,
  align.cgroup = NULL,
  css.rgroup = NULL,
  css.rgroup.sep = NULL,
  css.tspanner = NULL,
```

```
  css.tspanner.sep = NULL,
  css.total = NULL,
  css.cell = NULL,
  css.cgroup = NULL,
  css.header = NULL,
  css.header.border_bottom = NULL,
  css.class = NULL,
  css.table = NULL,
  pos.rowlabel = NULL,
  pos.caption = NULL,
  col.rgroup = NULL,
  col.columns = NULL,
  padding.rgroup = NULL,
  padding.tspanner = NULL,
  spacer.celltype = NULL,
  spacer.css.cgroup.bottom.border = NULL,
  spacer.css = NULL,
  spacer.content = NULL
)

appendHtmlTableStyle(
  x,
  align = NULL,
  align.header = NULL,
  align.cgroup = NULL,
  css.rgroup = NULL,
  css.rgroup.sep = NULL,
  css.tspanner = NULL,
  css.tspanner.sep = NULL,
  css.total = NULL,
  css.cell = NULL,
  css.cgroup = NULL,
  css.header = NULL,
  css.header.border_bottom = NULL,
  css.class = NULL,
  css.table = NULL,
  pos.rowlabel = NULL,
  pos.caption = NULL,
  col.rgroup = NULL,
  col.columns = NULL,
  padding.rgroup = NULL,
  padding.tspanner = NULL,
  spacer.celltype = NULL,
  spacer.css.cgroup.bottom.border = NULL,
  spacer.css = NULL,
  spacer.content = NULL
)
```

Arguments

x	The object that you later want to pass into htmlTable().
align	A character strings specifying column alignments, defaulting to 'c' to center. Valid chars for alignments are l = left, c = center and r = right. You can also specify align='c|c' and other LaTeX tabular formatting. If you want to set the alignment of the rownames this string needst to be ncol(x) + 1, otherwise it automatically pads the string with a left alignment for the rownames.
align.header	A character strings specifying alignment for column header, defaulting to centered, i.e. [paste][base::paste](rep('c',ncol(x)),collapse='').
align.cgroup	The justification of the cgroups
css.rgroup	CSS style for the rgroup, if different styles are wanted for each of the rgroups you can just specify a vector with the number of elements.
css.rgroup.sep	The line between different rgroups. The line is set to the TR element of the lower rgroup, i.e. you have to set the border-top/padding-top etc to a line with the expected function. This is only used for rgroups that are printed. You can specify different separators if you give a vector of rgroup - 1 length (this is since the first rgroup doesn't have a separator).
css.tspanner	The CSS style for the table spanner.
css.tspanner.sep	The line between different spanners.
css.total	The css of the total row if such is activated.
css.cell	The css.cell element allows you to add any possible CSS style to your table cells. See section below for details.
css.cgroup	The same as css.class but for cgroup formatting.
css.header	The header style, not including the cgroup style
css.header.border_bottom	The header bottom-border style, e.g. border-bottom: 1px solid grey
css.class	The html CSS class for the table. This allows directing html formatting through CSS directly at all instances of that class. *Note:* unfortunately the CSS is frequently ignored by word processors. This option is mostly inteded for web-presentations.
css.table	You can specify the the style of the table-element using this parameter
pos.rowlabel	Where the rowlabel should be positioned. This value can be "top", "bottom", "header", or a integer between 1 and nrow(cgroup) + 1. The options "bottom" and "header" are the same, where the row label is presented at the same level as the header.
pos.caption	Set to "bottom" to position a caption below the table instead of the default of "top".
col.rgroup	Alternating colors (zebra striping/banded rows) for each rgroup; one or two colors is recommended and will be recycled.
col.columns	Alternating colors for each column.
padding.rgroup	Generally two non-breakings spaces, i.e. , but some journals only have a bold face for the rgroup and leaves the subelements unindented.

`padding.tspanner`
: The table spanner is usually without padding but you may specify padding similar to `padding.rgroup` and it will be added to all elements, including the rgroup elements. This allows for a 3-level hierarchy if needed.

`spacer.celltype`
: When using cgroup the table headers are separated through a empty HTML cell that is by default filled with   (no-breaking-space) that prevents the cell from collapsing. The purpose of this is to prevent the headers underline to bleed into one as the underline is for the entire cell. You can alter this behavior by changing this option, valid options are `single_empty`, `skip`, `double_cell`. The `single_empty` is the default, the `skip` lets the header bleed into one and skips entirely, `double_cell` is for having two cells so that a vertical border ends up centered (specified using the `align` option). The arguments are matched internally using base::match.arg so you can specify only a part of the name, e.g. `"sk"` will match `"skip"`.

`spacer.css.cgroup.bottom.border`
: Defaults to none and used for separating cgroup headers. Due to a browser bug this is sometimes ignored and you may therefore need to set this to 1px solid white to enforce a white border.

`spacer.css`
: If you want the spacer cells to share settings you can set it here

`spacer.content`
: Defaults to   as this guarantees that the cell is not collapsed and is highly compatible when copy-pasting to word processors.

Details

The function stores the current theme (see `setHtmlTableTheme()`) + custom styles to the provided object as an `base::attributes()`. It is stored under the element `htmlTable.style` in the form of a list object.

Value

`x` with the style added as an attribute that the htmlTable then can use for formatting.

The `css.cell` argument

The `css.cell` parameter allows you to add any possible CSS style to your table cells. `css.cell` can be either a vector or a matrix.

If `css.cell` is a *vector*, it's assumed that the styles should be repeated throughout the rows (that is, each element in css.cell specifies the style for a whole column of 'x').

In the case of `css.cell` being a *matrix* of the same size of the `x` argument, each element of `x` gets the style from the corresponding element in css.cell. Additionally, the number of rows of `css.cell` can be `nrow(x) + 1` so the first row of of `css.cell` specifies the style for the header of `x`; also the number of columns of `css.cell` can be `ncol(x) + 1` to include the specification of style for row names of `x`.

Note that the `text-align` CSS field in the `css.cell` argument will be overriden by the `align` argument.

Excel has a specific css-style, `mso-number-format` that can be used for improving the copy-paste functionality. E.g. the style could be written as: css_matrix <- matrix(data = "mso-number-format:\"\\@\"", nrow = nrow(df), ncol = ncol(df))

See Also

Other htmlTableStyle: `hasHtmlTableStyle()`

Examples

```
library(magrittr)
matrix(1:4, ncol = 2) %>%
  addHtmlTableStyle(align = "c", css.cell = "background-color: orange;") %>%
  htmlTable(caption = "A simple style example")
```

---

concatHtmlTables *Function for concatenating* `htmlTable()`*s*

Description

Function for concatenating `htmlTable()`s

Usage

```
concatHtmlTables(tables, headers = NULL)
```

Arguments

tables	A list of `htmlTable()`s to be concatenated
headers	Either a string or a vector of strings that function as a header for each table. If none is provided it will use the names of the table list or a numeric number.

Value

`htmlTable()` class object

Examples

```
library(magrittr)

# Basic example
output <- matrix(1:4,
                 ncol=2,
                 dimnames = list(list("Row 1", "Row 2"),
                                 list("Column 1", "Column 2")))
htmlTable(output)

##########################################
# Below saves all outputs to a list that   #
```

```
# it outputted all at once at the end      #
# this is mostly for allowing you to view  #
# and evaluate each example section as     #
# they would otherwise be overwritten by   #
# eachother                                #
############################################
all_tables <- list()
htmlTable(output) ->
  all_tables[["Basic table"]]

# An advanced output
output <-
  matrix(ncol=6, nrow=8)

for (nr in 1:nrow(output)){
  for (nc in 1:ncol(output)){
    output[nr, nc] <-
      paste0(nr, ":", nc)
  }
}

output %>%
  addHtmlTableStyle(align="r",
                    col.columns = c(rep("none", 2),
                                    rep("#F5FBFF", 4)),
                    col.rgroup = c("none", "#F7F7F7"),
                    css.cell = "padding-left: .5em; padding-right: .2em;") %>%
  htmlTable(header =  paste(c("1st", "2nd",
                              "3rd", "4th",
                              "5th", "6th"),
                            "hdr"),
            rnames = paste(c("1st", "2nd",
                             "3rd",
                             paste0(4:8, "th")),
                           "row"),
            rgroup = paste("Group", LETTERS[1:3]),
            n.rgroup = c(2,4,nrow(output) - 6),
            cgroup = rbind(c("", "Column spanners", NA),
                           c("", "Cgroup 1", "Cgroup 2&dagger;")),
            n.cgroup = rbind(c(1,2,NA),
                             c(2,2,2)),
            caption="Basic table with both column spanners (groups) and row groups",
            tfoot="&dagger; A table footer commment",
            cspan.rgroup = 2) ->
  all_tables[["Advanced table"]]

# An advanced empty table
suppressWarnings({
  matrix(ncol = 6,
         nrow = 0) %>%
    addHtmlTableStyle(col.columns = c(rep("none", 2),
                                      rep("#F5FBFF", 4)),
                      col.rgroup = c("none", "#F7F7F7"),
```

```
                    css.cell = "padding-left: .5em; padding-right: .2em;") %>%
    htmlTable(align="r",
              header =  paste(c("1st", "2nd",
                                "3rd", "4th",
                                "5th", "6th"),
                              "hdr"),
              cgroup = rbind(c("", "Column spanners", NA),
                             c("", "Cgroup 1", "Cgroup 2&dagger;")),
              n.cgroup = rbind(c(1,2,NA),
                               c(2,2,2)),
            caption="Basic empty table with column spanners (groups) and ignored row colors",
              tfoot="&dagger; A table footer commment",
              cspan.rgroup = 2) ->
    all_tables[["Empty table"]]
})

# An example of how to use the css.cell for header styling
simple_output <- matrix(1:4, ncol=2)

simple_output %>%
  addHtmlTableStyle(css.cell = rbind(rep("background: lightgrey; font-size: 2em;",
                                         times=ncol(simple_output)),
                                     matrix("",
                                            ncol=ncol(simple_output),
                                            nrow=nrow(simple_output)))) %>%
  htmlTable(header = LETTERS[1:2]) ->
  all_tables[["Header formatting"]]

concatHtmlTables(all_tables)
# See vignette("tables", package = "htmlTable")
# for more examples
```

`getHtmlTableStyle` *Get style options for object*

Description

A wrap around the `base::attr()` that retrieves the style attribute used by `htmlTable()` (`htmlTable.style`).

Usage

```
getHtmlTableStyle(x)
```

Arguments

x	The object intended for `htmlTable()`.

Value

A `list` if the attribute exists, otherwise `NULL`

Examples

```
library(magrittr)

mx <- matrix(1:4, ncol = 2)
colnames(mx) <- LETTERS[1:2]
mx %>%
  addHtmlTableStyle(align = "l|r") %>%
  getHtmlTableStyle()
```

`getHtmlTableTheme` *Retrieve the* `htmlTable()` *theme list*

Description

A wrapper for a `getOption("htmlTable.theme")()` call that returns the standard theme unless one is set.

Usage

```
getHtmlTableTheme()
```

Value

`list` with the styles to be applied to the table

Examples

```
getHtmlTableTheme()
```

`hasHtmlTableStyle` *Check if object has a style set to it*

Description

If the attribute `htmlTable.style` is set it will check if the `style_name` exists and return a `logical`.

Usage

```
hasHtmlTableStyle(x, style_name)
```

Arguments

`x`	The object intended for `htmlTable()`.
`style_name`	A string that contains the style name.

Value

`logical` `TRUE` if the attribute and style is not `NULL`

See Also

Other htmlTableStyle: `addHtmlTableStyle()`

Examples

```
library(magrittr)

mx <- matrix(1:4, ncol = 2)
colnames(mx) <- LETTERS[1:2]
mx %>%
  addHtmlTableStyle(align = "l|r") %>%
  hasHtmlTableStyle("align")
```

---

htmlTable *Output an HTML table*

Description

This is a function for outputting a more advanced tables using HTML. The core philosophy is to bring column and row groups into the table and allow for a dense representation of complex tables. The HTML-output is designed for maximum compatibility with copy-paste functionality into word-processors. For adding styles, see `addHtmlTableStyle()` and themes `setHtmlTableTheme()`. *Note:* If you are using **tidyverse** and **dplyr** you may want to check out `tidyHtmlTable()` that automates many of the arguments that `htmlTable` requires.

Usage

```
htmlTable(
  x,
  header = NULL,
  rnames = NULL,
  rowlabel = NULL,
  caption = NULL,
  tfoot = NULL,
  label = NULL,
  rgroup = NULL,
  n.rgroup = NULL,
  cgroup = NULL,
  n.cgroup = NULL,
  tspanner = NULL,
  n.tspanner = NULL,
  total = NULL,
  ctable = TRUE,
```

```
  compatibility = getOption("htmlTableCompat", "LibreOffice"),
  cspan.rgroup = "all",
  escape.html = FALSE,
  ...
)

## Default S3 method:
htmlTable(
  x,
  header = NULL,
  rnames = NULL,
  rowlabel = NULL,
  caption = NULL,
  tfoot = NULL,
  label = NULL,
  rgroup = NULL,
  n.rgroup = NULL,
  cgroup = NULL,
  n.cgroup = NULL,
  tspanner = NULL,
  n.tspanner = NULL,
  total = NULL,
  ctable = TRUE,
  compatibility = getOption("htmlTableCompat", "LibreOffice"),
  cspan.rgroup = "all",
  escape.html = FALSE,
  ...
)

## S3 method for class 'htmlTable'
knit_print(x, ...)

## S3 method for class 'htmlTable'
print(x, useViewer, ...)
```

Arguments

x	The matrix/data.frame with the data. For the `print` and `knit_print` it takes a string of the class `htmlTable` as `x` argument.
header	A vector of character strings specifying column header, defaulting to `colnames(x)`
rnames	Default row names are generated from `rownames(x)`. If you provide `FALSE` then it will skip the row names. *Note:* For `data.frames` if you do `rownames(my_dataframe) <-NULL` it still has row names. Thus you need to use `FALSE` if you want to supress row names for `data.frames`.
rowlabel	If the table has row names or `rnames`, `rowlabel` is a character string containing the column heading for the `rnames`.
caption	Adds a table caption.

`tfoot` Adds a table footer (uses the <tfoot> HTML element). The output is run through `txtMergeLines()` simplifying the generation of multiple lines.

`label` A text string representing a symbolic label for the table for referencing as an anchor. All you need to do is to reference the table, for instance see table 2. This is known as the element's id attribute, i.e. table id, in HTML linguo, and should be unique id for an HTML element in contrast to the `css.class` element attribute.

`rgroup` A vector of character strings containing headings for row groups. `n.rgroup` must be present when `rgroup` is given. See detailed description in section below.

`n.rgroup` An integer vector giving the number of rows in each grouping. If `rgroup` is not specified, `n.rgroup` is just used to divide off blocks of rows by horizontal lines. If `rgroup` is given but `n.rgroup` is omitted, `n.rgroup` will default so that each row group contains the same number of rows. If you want additional rgroup column elements to the cells you can sett the "add" attribute to `rgroup` through `attr(rgroup,"add")`, see below explaining section.

`cgroup` A vector, matrix or list of character strings defining major column header. The default is to have none. These elements are also known as *column spanners*. If you want a column *not* to have a spanner then put that column as "". If you pass cgroup and `n.crgroup` as matrices you can have column spanners for several rows. See cgroup section below for details.

`n.cgroup` An integer vector, matrix or list containing the number of columns for which each element in cgroup is a heading. For example, specify `cgroup=c("Major_1","Major_2")`, `n.cgroup=c(3,3)` if `"Major_1"` is to span columns 1-3 and `"Major_2"` is to span columns 4-6. `rowlabel` does not count in the column numbers. You can omit `n.cgroup` if all groups have the same number of columns. If the `n.cgroup` is one less than the number of columns in the matrix/data.frame then it automatically adds those.

`tspanner` The table spanner is somewhat of a table header that you can use when you want to join different tables with the same columns.

`n.tspanner` An integer vector with the number of rows or `rgroups` in the original matrix that the table spanner should span. If you have provided one fewer n.tspanner elements the last will be imputed from the number of `rgroups` (if you have provided `rgroup` and `sum(n.tspanner) < length(rgroup)`) or the number of rows in the table.

`total` The last row is sometimes a row total with a border on top and bold fonts. Set this to `TRUE` if you are interested in such a row. If you want a total row at the end of each table spanner you can set this to `"tspanner"`.

`ctable` If the table should have a double top border or a single a' la LaTeX ctable style

`compatibility` Is default set to `LibreOffice` as some settings need to be in old HTML format as Libre Office can't handle some commands such as the css caption-alignment. Note: this option is not yet fully implemented for all details, in the future I aim to generate a HTML-correct table and one that is aimed at Libre Office compatibility. Word-compatibility is difficult as Word ignores most settings and destroys all layout attempts (at least that is how my 2010 version behaves). You can additinally use the `options(htmlTableCompat = "html")` if you want a change to apply to the entire document. MS Excel sometimes misinterprets

certain cell data when opening HTML-tables (eg. 1/2 becomes 1. February). To avoid this please specify the correct Microsoft Office format for each cell in the table using the css.cell-argument. To make MS Excel interpret everything as text use "mso-number-format:\"\@\"".

`cspan.rgroup` The number of columns that an `rgroup` should span. It spans by default all columns but you may want to limit this if you have column colors that you want to retain.

`escape.html` logical: should HTML characters be escaped? Defaults to FALSE.

`...` Passed on to `print.htmlTable` function and any argument except the `useViewer` will be passed on to the `base::cat()` functions arguments. *Note:* as of version 2.0.0 styling options are still allowed but it is recommended to instead preprocess your object with `addHtmlTableStyle()`.

`useViewer` If you are using RStudio there is a viewer thar can render the table within that is envoced if in `base::interactive()` mode. Set this to `FALSE` if you want to remove that functionality. You can also force the function to call a specific viewer by setting this to a viewer function, e.g. `useViewer = utils::browseURL` if you want to override the default RStudio viewer. Another option that does the same is to set the `options(viewer=utils::browseURL)` and it will default to that particular viewer (this is how RStudio decides on a viewer). *Note:* If you want to force all output to go through the `base::cat()` the set [options][base::options](htmlTable.cat = TRUE).

Value

`string` Returns a string of class `htmlTable`

Multiple rows of column spanners `cgroup`

If you want to have a column spanner in multiple levels you can set the `cgroup` and `n.cgroup` arguments to a `matrix` or `list`.

If the different levels have different number of elements and you have provided a *matrix* you need to set the ones that lack elements to NA. For instance `cgroup = rbind(c("first","second",NA),c("a","b","c"))`. And the corresponding `n.cgroup` would be `n.cgroup = rbind(c(1,2,NA),c(2,1,2))`. for a table consisting of 5 columns. The "first" spans the first two columns, the "second" spans the last three columns, "a" spans the first two, "b" the middle column, and "c" the last two columns.

It is recommended to use `list` as you will not have to bother with the `NA`.

If you want leave a `cgroup` empty then simply provide `""` as the `cgroup`.

The `rgroup` argument

The `rgroup` allows you to smoothly group rows. Each row within a group receives an indention of two blank spaces and are grouped with their corresponding `rgroup` element. The `sum(n.rgroup)` should always be equal or less than the matrix rows. If less then it will pad the remaining rows with either an empty `rgroup`, i.e. an "" or if the `rgroup` is one longer than the `n.rgroup` the last `n.rgroup` element will be calculated through `nrow(x) -sum(n.rgroup)` in order to make the table generating smoother.

The add attribute to rgroup

You can now have an additional element at the rgroup level by specifying the attr(rgroup,'add'). The value can either be a vector, a list, or a matrix. See vignette("general",package = "htmlTable") for examples.

- A vector of either equal number of rgroups to the number of rgroups that aren't empty, i.e. rgroup[rgroup != ""]. Or a named vector where the name must correspond to either an rgroup or to an rgroup number.
- A list that has exactly the same requirements as the vector. In addition to the previous we can also have a list with column numbers within as names within the list.
- A matrix with the dimension nrow(x) x ncol(x) or nrow(x) x 1 where the latter is equivalent to a named vector. If you have rownames these will resolve similarly to the names to the list/vector arguments. The same thing applies to colnames.

Important knitr-note

This function will only work with **knitr** outputting *HTML*, i.e. markdown mode. As the function returns raw HTML-code the compatibility with non-HTML formatting is limited, even with pandoc.

Thanks to the the knitr::knit_print() and the knitr::asis_output() the results='asis' is *no longer needed* except within for-loops. If you have a knitr-chunk with a for loop and use print() to produce raw HTML you must set the chunk option results='asis'. *Note*: the print-function relies on the base::interactive() function for determining if the output should be sent to a browser or to the terminal. In vignettes and other directly knitted documents you may need to either set useViewer = FALSE alternatively set options(htmlTable.cat = TRUE).

RStudio's notebook

RStudio has an interactive notebook that allows output directly into the document. In order for the output to be properly formatted it needs to have the class of html. The htmlTable tries to identify if the environment is a notebook document (uses the **rstudioapi** and identifies if its a file with and Rmd file ending or if there is an element with html_notebook). If you don't want this behavior you can remove it using the options(htmlTable.skip_notebook = TRUE).

Table counter

If you set the option table_counter you will get a Table 1,2,3 etc before each table, just set options(table_counter=TRUE). If you set it to a number then that number will correspond to the start of the table_counter. The table_counter option will also contain the number of the last table, this can be useful when referencing it in text. By setting the option options(table_counter_str = "Table %s: ") you can manipulate the counter table text that is added prior to the actual caption. Note, you should use the sprintf() %s instead of %d as the software converts all numbers to characters for compatibility reasons. If you set options(table_counter_roman = TRUE) then the table counter will use Roman numerals instead of Arabic.

Empty data frames

An empty data frame will result in a warning and output an empty table, provided that rgroup and n.rgroup are not specified. All other row layout options will be ignored.

Options

There are multiple options that can be set, here is a set of the perhaps most used

- `table_counter` - logical - activates a counter for each table
- `table_counter_roman` - logical - if true the counter is in Roman numbers, i.e. I, II, III, IV...
- `table_counter_str` - string - the string used for generating the table counter text
- `useViewer` - logical - if viewer should be used fro printing the table
- `htmlTable.cat` - logical - if the output should be directly sent to `cat()`
- `htmlTable.skip_notebook` - logical - skips the logic for detecting notebook
- `htmlTable.pretty_indentation` - logical - there was some issues in previous Pandoc versions where HTML indentation caused everything to be interpreted as code. This seems to be fixed and if you want to look at the raw HTML code it is nice to have this set to `TRUE` so that the tags and elements are properly indented.
- `htmlTableCompat` - string - see parameter description

Other

Copy-pasting: As you copy-paste results into Word you need to keep the original formatting. Either right click and choose that paste option or click on the icon appearing after a paste. Currently the following compatibilities have been tested with MS Word 2016:

- **Internet Explorer** (v. 11.20.10586.0) Works perfectly when copy-pasting into Word
- **RStudio** (v. 0.99.448) Works perfectly when copy-pasting into Word. *Note:* can have issues with multi-line `cgroups` - see bug
- **Chrome** (v. 47.0.2526.106) Works perfectly when copy-pasting into Word. *Note:* can have issues with multi-line `cgroups` - see bug
- **Firefox** (v. 43.0.3) Works poorly - looses font-styling, lines and general feel
- **Edge** (v. 25.10586.0.0) Works poorly - looses lines and general feel

Direct word processor opening: Opening directly in Libre Office or Word is no longer recommended. You get much prettier results using the cut-and-paste option.

Google docs: Copy-paste directly into a Google docs document is handled rather well. This seems to work especially well when the paste comes directly from a Chrome browser.

Note that when using complex `cgroup` alignments with multiple levels not every browser is able to handle this. For instance the RStudio webkit browser seems to have issues with this and a bug has been filed.

As the table uses HTML for rendering you need to be aware of that headers, row names, and cell values should try respect this for optimal display. Browsers try to compensate and frequently the tables still turn out fine but it is not advised. Most importantly you should try to use < instead of < and > instead of >. You can find a complete list of HTML characters here.

Lastly, I want to mention that function was inspired by the `Hmisc::latex()` that can be an excellent alternative if you wish to switch to PDF-output. For the sibling function `tidyHtmlTable()` you can directly switch between the two using the `table_fn` argument.

See Also

addHtmlTableStyle(), setHtmlTableTheme(), tidyHtmlTable(). txtMergeLines(), Hmisc::latex()

Other table functions: tblNoLast(), tblNoNext()

Examples

```
library(magrittr)

# Basic example
output <- matrix(1:4,
                 ncol=2,
                 dimnames = list(list("Row 1", "Row 2"),
                                 list("Column 1", "Column 2")))
htmlTable(output)

##########################################
# Below saves all outputs to a list that #
# it outputted all at once at the end    #
# this is mostly for allowing you to view #
# and evaluate each example section as   #
# they would otherwise be overwritten by #
# eachother                              #
##########################################
all_tables <- list()
htmlTable(output) ->
  all_tables[["Basic table"]]

# An advanced output
output <-
  matrix(ncol=6, nrow=8)

for (nr in 1:nrow(output)){
  for (nc in 1:ncol(output)){
    output[nr, nc] <-
      paste0(nr, ":", nc)
  }
}

output %>%
  addHtmlTableStyle(align="r",
                    col.columns = c(rep("none", 2),
                                    rep("#F5FBFF", 4)),
                    col.rgroup = c("none", "#F7F7F7"),
                    css.cell = "padding-left: .5em; padding-right: .2em;") %>%
  htmlTable(header =  paste(c("1st", "2nd",
                              "3rd", "4th",
                              "5th", "6th"),
                            "hdr"),
            rnames = paste(c("1st", "2nd",
                             "3rd",
                             paste0(4:8, "th")),
                           "row"),
```

```
            rgroup = paste("Group", LETTERS[1:3]),
            n.rgroup = c(2,4,nrow(output) - 6),
            cgroup = rbind(c("", "Column spanners", NA),
                           c("", "Cgroup 1", "Cgroup 2&dagger;")),
            n.cgroup = rbind(c(1,2,NA),
                             c(2,2,2)),
            caption="Basic table with both column spanners (groups) and row groups",
            tfoot="&dagger; A table footer commment",
            cspan.rgroup = 2) ->
  all_tables[["Advanced table"]]

# An advanced empty table
suppressWarnings({
  matrix(ncol = 6,
         nrow = 0) %>%
    addHtmlTableStyle(col.columns = c(rep("none", 2),
                                      rep("#F5FBFF", 4)),
                      col.rgroup = c("none", "#F7F7F7"),
                      css.cell = "padding-left: .5em; padding-right: .2em;") %>%
    htmlTable(align="r",
              header =  paste(c("1st", "2nd",
                                "3rd", "4th",
                                "5th", "6th"),
                              "hdr"),
              cgroup = rbind(c("", "Column spanners", NA),
                             c("", "Cgroup 1", "Cgroup 2&dagger;")),
              n.cgroup = rbind(c(1,2,NA),
                               c(2,2,2)),
          caption="Basic empty table with column spanners (groups) and ignored row colors",
              tfoot="&dagger; A table footer commment",
              cspan.rgroup = 2) ->
    all_tables[["Empty table"]]
})

# An example of how to use the css.cell for header styling
simple_output <- matrix(1:4, ncol=2)

simple_output %>%
  addHtmlTableStyle(css.cell = rbind(rep("background: lightgrey; font-size: 2em;",
                                         times=ncol(simple_output)),
                                     matrix("",
                                            ncol=ncol(simple_output),
                                            nrow=nrow(simple_output)))) %>%
  htmlTable(header = LETTERS[1:2]) ->
  all_tables[["Header formatting"]]

concatHtmlTables(all_tables)
# See vignette("tables", package = "htmlTable")
# for more examples
```

---

htmlTableWidget *htmlTable with pagination widget*

Description

This widget renders a table with pagination into an htmlwidget

Usage

```
htmlTableWidget(
  x,
  number_of_entries = c(10, 25, 100),
  width = NULL,
  height = NULL,
  elementId = NULL,
  ...
)
```

Arguments

x	A data frame to be rendered
number_of_entries	a numeric vector with the number of entries per page to show. If there is more than one number given, the user will be able to show the number of rows per page in the table.
width	Fixed width for widget (in css units). The default is NULL, which results in intelligent automatic sizing based on the widget's container.
height	Fixed height for widget (in css units). The default is NULL, which results in intelligent automatic sizing based on the widget's container.
elementId	Use an explicit element ID for the widget (rather than an automatically generated one). Useful if you have other JavaScript that needs to explicitly discover and interact with a specific widget instance.
...	Additional parameters passed to htmlTable

Value

an htmlwidget showing the paginated table

---

htmlTableWidget-shiny *Shiny bindings for htmlTableWidget*

Description

Output and render functions for using htmlTableWidget within Shiny applications and interactive Rmd documents.

Usage

```
htmlTableWidgetOutput(outputId, width = "100%", height = "400px")

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

Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.
expr	An expression that generates a htmlTableWidget()
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

Examples

```
## Not run:
# In the UI:
htmlTableWidgetOutput("mywidget")
# In the server:
renderHtmlTableWidget({
  htmlTableWidget(iris)
})

## End(Not run)
```

---

innerJoinByCommonCols *A simple function for joining two tables by their intersected columns*

Description

A simple function for joining two tables by their intersected columns

Usage

```
innerJoinByCommonCols(x, y)
```

Arguments

x	data.frame
y	data.frame

Value

data.frame

`interactiveTable` *An interactive table that allows you to limit the size of boxes*

Description

This function wraps the htmlTable and adds JavaScript code for toggling the amount of text shown in any particular cell.

Usage

```
interactiveTable(
  x,
  ...,
  txt.maxlen = 20,
  button = getOption("htmlTable.interactiveTable.button", default = FALSE),
  minimized.columns,
  js.scripts = c()
)

## S3 method for class 'htmlTable'
interactiveTable(
  tbl,
  txt.maxlen = 20,
  button = getOption("htmlTable.interactiveTable.button", default = FALSE),
  minimized.columns = NULL,
  js.scripts = c()
)

## S3 method for class 'interactiveTable'
knit_print(x, ...)

## S3 method for class 'interactiveTable'
print(x, useViewer, ...)
```

Arguments

`x`	The interactive table that is to be printed
`...`	The exact same parameters as `htmlTable()` uses
`txt.maxlen`	The maximum length of a text
`button`	Indicator if the cell should be clickable or if a button should appear with a plus/minus
`minimized.columns`	Notifies if any particular columns should be collapsed from start
`js.scripts`	If you want to add your own JavaScript code you can just add it here. All code is merged into one string where each section is wrapped in it's own <scrip></script> element.

`tbl`	An htmlTable object can be directly passed into the function
`useViewer`	If you are using RStudio there is a viewer thar can render the table within that is envoced if in `base::interactive()` mode. Set this to `FALSE` if you want to remove that functionality. You can also force the function to call a specific viewer by setting this to a viewer function, e.g. `useViewer = utils::browseURL` if you want to override the default RStudio viewer. Another option that does the same is to set the `options(viewer=utils::browseURL)` and it will default to that particular viewer (this is how RStudio decides on a viewer). *Note:* If you want to force all output to go through the `base::cat()` the set [options][base::options](htmlTable.cat = TRUE).

Value

An htmlTable with a javascript attribute containing the code that is then printed

Examples

```
library(magrittr)
# A simple output
long_txt <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi
ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit
in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur
sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt
mollit anim id est laborum"
short_txt <- gsub("(^[^.]+).*", "\\1", long_txt)

cbind(rep(short_txt, 2),
      rep(long_txt, 2)) %>%
  addHtmlTableStyle(col.rgroup = c("#FFF", "#EEF")) %>%
  interactiveTable(minimized.columns = ncol(.),
                   header = c("Short", "Long"),
                   rnames = c("First", "Second"))
```

---

prBindDataListIntoColumns

Merge columns into a tibble

Description

Almost the same as `tibble::tibble()` but it solves the issue with some of the arguments being columns and some just being vectors.

Usage

```
prBindDataListIntoColumns(dataList)
```

Arguments

dataList	list with the columns/data.frames

Value

data.frame object

---

prConvertDfFactors *Convert all factors to characters to print them as they expected*

Description

Convert all factors to characters to print them as they expected

Usage

```
prConvertDfFactors(x)
```

Arguments

x	The matrix/data.frame with the data. For the print and knit_print it takes a string of the class htmlTable as x argument.

Value

The data frame with factors as characters

---

prepGroupCounts *Retrieves counts for rgroup, cgroup, & tspanner arguments*

Description

This function is a wrapper to base::rle() that does exactly this but is a little too picky about input values.

Usage

```
prepGroupCounts(x)
```

Arguments

x	The vector to process

Value

list(n = rle$lengths, names = rle$values)

Examples

```
prepGroupCounts(c(1:3, 3:1))
```

`prEscapeHtml` *Remove html entities from table*

Description

Removes the htmlEntities from table input data. Note that this also replaces $ signs in order to remove the MathJax issue.

Usage

```
prEscapeHtml(x)
```

Arguments

`x`	The matrix/data.frame with the data. For the `print` and `knit_print` it takes a string of the class `htmlTable` as `x` argument.

Value

`x` without the html entities

See Also

Other hidden helper functions for htmlTable: `prAddCells()`, `prAddEmptySpacerCell()`, `prAddSemicolon2StrEnd()`, `prGetCgroupHeader()`, `prGetRowlabelPos()`, `prGetStyle()`, `prPrepInputMatrixDimensions()`, `prPrepareAlign()`, `prPrepareCgroup()`, `prTblNo()`

`prExtractElementsAndConvertToTbl`
Extract the elements and generate a table with unique elements

---

Description

Extract the elements and generate a table with unique elements

Usage

```
prExtractElementsAndConvertToTbl(x, elements)
```

Arguments

`x`	`list` with columns to be joined
`elements`	`char` vector with the elements to select

SCB *Average age in Sweden*

Description

For the vignettes there is a dataset downloaded by using the `get_pxweb_data()` call. The data is from SCB (Statistics Sweden) and downloaded using the pxweb package:

Author(s)

Max Gordon <max@gforge.se>

References

https://scb.se

Examples

```
## Not run:
# The data was generated through downloading via the API
library(pxweb)

# Get the last 15 years of data (the data always lags 1 year)
current_year <- as.integer(format(Sys.Date(), "%Y")) -1
SCB <- get_pxweb_data(
 url = "http://api.scb.se/OV0104/v1/doris/en/ssd/BE/BE0101/BE0101B/BefolkningMedelAlder",
  dims = list(Region = c('00', '01', '03', '25'),
              Kon = c('1', '2'),
              ContentsCode = c('BE0101G9'),
              Tid = (current_year-14):current_year),
  clean = TRUE)

# Some cleaning was needed before use
SCB$region <- factor(substring(as.character(SCB$region), 4))
Swe_ltrs <- c("å" = "&aring;",
              "Å" = "&Aring;",
              "ä" = "&auml;",
              "Ä" = "&Auml;",
              "ö" = "&ouml;",
              "Ö" = "&Ouml;")
for (i in 1:length(Swe_ltrs)){
  levels(SCB$region) <- gsub(names(Swe_ltrs)[i],
              Swe_ltrs[i],
              levels(SCB$region))
}

save(SCB, file = "data/SCB.rda")

## End(Not run)
```

setHtmlTableTheme *Set or update theme for* htmlTable()

Description

The theme guides many of the non-data objects visual appearance. The theme can be over-ridden by settings for each table. Too get a more complete understanding of the options, see addHtmlTableStyle().

Usage

```
setHtmlTableTheme(
  theme = NULL,
  align = NULL,
  align.header = NULL,
  align.cgroup = NULL,
  css.rgroup = NULL,
  css.rgroup.sep = NULL,
  css.tspanner = NULL,
  css.tspanner.sep = NULL,
  css.total = NULL,
  css.cell = NULL,
  css.cgroup = NULL,
  css.header = NULL,
  css.header.border_bottom = NULL,
  css.class = NULL,
  css.table = NULL,
  pos.rowlabel = NULL,
  pos.caption = NULL,
  col.rgroup = NULL,
  col.columns = NULL,
  padding.rgroup = NULL,
  padding.tspanner = NULL,
  spacer.celltype = NULL,
  spacer.css.cgroup.bottom.border = NULL,
  spacer.css = NULL,
  spacer.content = NULL
)
```

Arguments

theme	A list containing all the styles or a string that is matched to some of the preset style (See details below in the *Theme options* section). *Note*: the full name of the theme is not required as they are matched using base::match.arg().
align	A character strings specifying column alignments, defaulting to 'c' to center. Valid chars for alignments are l = left, c = center and r = right. You can also specify align='c|c' and other LaTeX tabular formatting. If you want to set

the alignment of the rownames this string needst to be `ncol(x) + 1`, otherwise it automatically pads the string with a left alignment for the rownames.

`align.header` A character strings specifying alignment for column header, defaulting to centered, i.e. [paste][base::paste](rep('c',ncol(x)),collapse='').

`align.cgroup` The justification of the `cgroups`

`css.rgroup` CSS style for the rgroup, if different styles are wanted for each of the rgroups you can just specify a vector with the number of elements.

`css.rgroup.sep` The line between different rgroups. The line is set to the TR element of the lower rgroup, i.e. you have to set the border-top/padding-top etc to a line with the expected function. This is only used for rgroups that are printed. You can specify different separators if you give a vector of rgroup - 1 length (this is since the first rgroup doesn't have a separator).

`css.tspanner` The CSS style for the table spanner.

`css.tspanner.sep`
The line between different spanners.

`css.total` The css of the total row if such is activated.

`css.cell` The css.cell element allows you to add any possible CSS style to your table cells. See section below for details.

`css.cgroup` The same as `css.class` but for cgroup formatting.

`css.header` The header style, not including the cgroup style

`css.header.border_bottom`
The header bottom-border style, e.g. border-bottom: 1px solid grey

`css.class` The html CSS class for the table. This allows directing html formatting through CSS directly at all instances of that class. *Note:* unfortunately the CSS is frequently ignored by word processors. This option is mostly inteded for web-presentations.

`css.table` You can specify the the style of the table-element using this parameter

`pos.rowlabel` Where the rowlabel should be positioned. This value can be `"top"`, `"bottom"`, `"header"`, or a integer between 1 and `nrow(cgroup) + 1`. The options `"bottom"` and `"header"` are the same, where the row label is presented at the same level as the header.

`pos.caption` Set to `"bottom"` to position a caption below the table instead of the default of `"top"`.

`col.rgroup` Alternating colors (zebra striping/banded rows) for each `rgroup`; one or two colors is recommended and will be recycled.

`col.columns` Alternating colors for each column.

`padding.rgroup` Generally two non-breakings spaces, i.e.   , but some journals only have a bold face for the rgroup and leaves the subelements unindented.

`padding.tspanner`
The table spanner is usually without padding but you may specify padding similar to `padding.rgroup` and it will be added to all elements, including the rgroup elements. This allows for a 3-level hierarchy if needed.

`spacer.celltype`
: When using cgroup the table headers are separated through a empty HTML cell that is by default filled with   (no-breaking-space) that prevents the cell from collapsing. The purpose of this is to prevent the headers underline to bleed into one as the underline is for the entire cell. You can alter this behavior by changing this option, valid options are `single_empty`, `skip`, `double_cell`. The `single_empty` is the default, the `skip` lets the header bleed into one and skips entirely, `double_cell` is for having two cells so that a vertical border ends up centered (specified using the `align` option). The arguments are matched internally using base::match.arg so you can specify only a part of the name, e.g. `"sk"` will match `"skip"`.

`spacer.css.cgroup.bottom.border`
: Defaults to `none` and used for separating cgroup headers. Due to a browser bug this is sometimes ignored and you may therefore need to set this to 1px solid white to enforce a white border.

`spacer.css`
: If you want the spacer cells to share settings you can set it here

`spacer.content`
: Defaults to   as this guarantees that the cell is not collapsed and is highly compatible when copy-pasting to word processors.

Value

An invisible `list` with the new theme

Theme options

The styles available are:

- `standard`: The traditional standard style used in `htmlTable()` since the early days
- Google docs: A style that is optimized for copy-pasting into documents on Google drive. This is geared towards minimal padding and margins so that the table is as dense as possible.
- `blank`: Just as the name suggests the style is completly empty in terms of CSS. Positions for rowlabel and caption are set to `bottom` as these cannot be blank.

You can also provide your own style. Each style should be a names vector, e.g. `c(width = "100px",color = "red")` or just a real css string, width: 100px; color: red;.

Examples

```
## Not run:
setHtmlTableTheme("Google", align = "r")

## End(Not run)
```

tblNoLast *Gets the last table number*

Description

The function relies on `options("table_counter")` in order to keep track of the last number.

Usage

```
tblNoLast(roman = getOption("table_counter_roman", FALSE))
```

Arguments

roman	Whether or not to use roman numbers instead of arabic. Can also be set through `options(table_caption_no_roman = TRUE)`

See Also

Other table functions: `htmlTable`, `tblNoNext()`

Examples

```
org_opts <- options(table_counter=1)
tblNoLast()
options(org_opts)
```

tblNoNext *Gets the next table number*

Description

The function relies on `options("table_counter")` in order to keep track of the last number.

Usage

```
tblNoNext(roman = getOption("table_counter_roman", FALSE))
```

Arguments

roman	Whether or not to use roman numbers instead of arabic. Can also be set through `options(table_caption_no_roman = TRUE)`

See Also

Other table functions: `htmlTable`, `tblNoLast()`

Examples

```
org_opts <- options(table_counter=1)
tblNoNext()
options(org_opts)
```

`tidyHtmlTable` *Generate an htmlTable using tidy data as input*

Description

Builds an `htmlTable` by mapping columns from the input data, `x`, to elements of an output `htmlTable` (e.g. `rnames`, `header`, etc.). This provides a **ggplot2**-like interface you can pivot rows/columns as required. The typical use case is when you are using `dplyr` together with the `tidyverse` data processing functions, see `vignette("tidyHtmlTable")`.

Usage

```
tidyHtmlTable(
  x,
  value,
  header,
  rnames,
  rgroup,
  hidden_rgroup,
  cgroup,
  tspanner,
  hidden_tspanner,
  skip_removal_warning = getOption("htmlTable.skip_removal_warning", FALSE),
  table_fn = htmlTable,
  ...
)
```

Arguments

`x`	Tidy data used to build the `htmlTable`
`value`	The column containing values filling individual cells of the output `htmlTable`. Defaults to "value" as used by `tidyr::pivot_longer()`.
`header`	The column in `x` specifying column headings
`rnames`	The column in `x` specifying row names. Defaults to "name" as used by `tidyr::pivot_longer()`.
`rgroup`	The column in `x` specifying row groups
`hidden_rgroup`	`strings` with `rgroup` values that will be hidden (the values will still be there but the spanner will be set to "" and thus ignored by `htmlTable()`).
`cgroup`	The column or columns in `x` specifying the column groups
`tspanner`	The column in `x` specifying `tspanner` groups

hidden_tspanner
: strings with tspanner values that will be hidden (the values will still be there but the spanner will be set to "" and thus ignored by htmlTable()).

skip_removal_warning
: boolean suppress warning message when removing NA columns.

table_fn
: The table function that should receive the input, defaults to htmlTable() but you can provide any function that uses the same input formatting. This package was inspired by the Hmisc::latex() function.

...
: Additional arguments that will be passed to the inner htmlTable() function

Value

Returns html code that will build a pretty table

Column-mapping parameters

The tidyHtmlTable function is designed to work like ggplot2 in that columns from x are mapped to specific parameters from the htmlTable function. At minimum, x must contain the names of columns mapping to rnames, header, and rnames. header and rnames retain the same meaning as in the htmlTable function. value contains the individual values that will be used to fill each cell within the output htmlTable.

A full list of parameters from htmlTable which may be mapped to columns within x include:

- value
- header
- rnames
- rgroup
- cgroup
- tspanner

Also note that the coordinates of each value within x must be unambiguously mapped to a position within the output htmlTable. Therefore, the each row-wise combination the variables specified above contained in x must be unique.

Sorting

Sorting of rows is as of version 2.0 skipped as we may have situations with repeating inputs and this can easily be performed pre-function by calling dplyr::arrange() prior to tidyHtmlTable.

Columns are sorted by arrange(cgroup,header) where cgroup will be expanded to the columns of the cgroup argument, e.g. cgroup = c(a, b), header = c will become arrange(a,b,c). If you want to sort in non-alphabetic order you can provide a factor variable and that information will be retained.

Hidden values

`htmlTable` Allows for some values within `rgroup`, `cgroup`, etc. to be specified as `""`. The following parameters allow for specific values to be treated as if they were a string of length zero in the `htmlTable` function.

- `hidden_rgroup`
- `hidden_tspanner`

Simple tibble output

The tibble discourages the use of row names. There is therefore a convenience option for `tidyHtmlTable` where you can use the function just as you would with `htmlTable()` where `rnames` is populated with the `rnames` argument provided using `tidyselect` syntax (defaults to the "names" column if present int the input data).

Additional dependencies

In order to run this function you also must have **dplyr**, **tidyr**, **tidyselect** and **purrr** packages installed. These have been removed due to the additional 20 Mb that these dependencies added (issue #47). *Note:* if you use **tidyverse** it will already have all of these and you do not need to worry.

See Also

`htmlTable()`

Examples

```
library(tibble)
library(dplyr)
library(tidyr)

mtcars %>%
  rownames_to_column() %>%
  select(rowname, cyl, gear, hp, mpg, qsec) %>%
  pivot_longer(names_to = "per_metric",
               cols = c(hp, mpg, qsec)) %>%
  group_by(cyl, gear, per_metric) %>%
  summarise(
    Mean = round(mean(value), 1),
    SD = round(sd(value), 1),
    Min = round(min(value), 1),
    Max = round(max(value), 1)
  ) %>%
  pivot_longer(names_to = "summary_stat",
               cols = c(Mean, SD, Min, Max)) %>%
  ungroup() %>%
  mutate(
    gear = paste(gear, "Gears"),
    cyl = paste(cyl, "Cylinders")
  ) %>%
  addHtmlTableStyle(align = "r") %>%
```

```
tidyHtmlTable(
  header = gear,
  cgroup = cyl,
  rnames = summary_stat,
  rgroup = per_metric,
  skip_removal_warning = TRUE)
```

---

txtInt *SI or English formatting of an integer*

Description

English uses ',' between every 3 numbers while the SI format recommends a ' ' if x > 10^4. The scientific form 10e+? is furthermore avoided.

Usage

```
txtInt(x, language = "en", html = TRUE, ...)
```

Arguments

x	The integer variable
language	The ISO-639-1 two-letter code for the language of interest. Currently only English is distinguished from the ISO format using a ',' as the separator.
html	If the format is used in HTML context then the space should be a non-breaking space,
...	Passed to base::format()

Value

string

See Also

Other text formatters: txtMergeLines(), txtPval(), txtRound()

Examples

```
txtInt(123)

# Supplying a matrix
txtInt(matrix(c(1234, 12345, 123456, 1234567), ncol = 2))

# Missing are returned as empty strings, i.e. ""
txtInt(c(NA, 1e7))
```

txtMergeLines *A merges lines while preserving the line break for HTML/LaTeX*

Description

This function helps you to do a table header with multiple lines in both HTML and in LaTeX. In HTML this isn't that tricky, you just use the
 command but in LaTeX I often find myself writing vbox/hbox stuff and therefore I've created this simple helper function

Usage

```
txtMergeLines(..., html = 5)
```

Arguments

`...`	The lines that you want to be joined
`html`	If HTML compatible output should be used. If `FALSE` it outputs LaTeX formatting. Note if you set this to 5 then the HTML5 version of *br* will be used: otherwise it uses the that is compatible with the XHTML-formatting.

Value

string

See Also

Other text formatters: `txtInt()`, `txtPval()`, `txtRound()`

Examples

```
txtMergeLines("hello", "world")
txtMergeLines("hello", "world", html=FALSE)
txtMergeLines("hello", "world", list("A list", "is OK"))
```

txtPval *Formats the p-values*

Description

Gets formatted p-values. For instance you often want `0.1234` to be `0.12` while also having two values up until a limit, i.e. `0.01234` should be `0.012` while `0.001234` should be `0.001`. Furthermore you want to have < 0.001 as it becomes ridiculous to report anything below that value.

Usage

```
txtPval(pvalues, lim.2dec = 10^-2, lim.sig = 10^-4, html = TRUE, ...)
```

Arguments

pvalues	The p-values
lim.2dec	The limit for showing two decimals. E.g. the p-value may be 0.056 and we may want to keep the two decimals in order to emphasize the proximity to the all-mighty 0.05 p-value and set this to 10^-2. This allows that a value of 0.0056 is rounded to 0.006 and this makes intuitive sense as the 0.0056 level as this is well below the 0.05 value and thus not as interesting to know the exact proximity to 0.05. *Disclaimer:* The 0.05-limit is really silly and debated, unfortunately it remains a standard and this package tries to adapt to the current standards in order to limit publication associated issues.
lim.sig	The significance limit for the less than sign, i.e. the '<'
html	If the less than sign should be < or < as needed for HTML output.
...	Currently only used for generating warnings of deprecated call parameters.

Value

vector

See Also

Other text formatters: txtInt(), txtMergeLines(), txtRound()

Examples

```
txtPval(c(0.10234,0.010234, 0.0010234, 0.000010234))
```

---

txtRound *A convenient rounding function*

Description

If you provide a string value in X the function will try to round this if a numeric text is present. If you want to skip certain rows/columns then use the excl.* arguments.

Usage

```
txtRound(x, ...)

## Default S3 method:
txtRound(
  x,
  digits = 0,
  digits.nonzero = NA,
  txt.NA = "",
  dec = getOption("htmlTable.decimal_marker", default = "."),
  scientific = NULL,
```

```
  txtInt_args = getOption("htmlTable.round_int", default = NULL),
  ...
)

## S3 method for class 'data.frame'
txtRound(x, ...)

## S3 method for class 'table'
txtRound(x, ...)

## S3 method for class 'matrix'
txtRound(x, digits = 0, excl.cols = NULL, excl.rows = NULL, ...)
```

Arguments

x	The value/vector/data.frame/matrix to be rounded
...	Passed to next method
digits	The number of digits to round each element to. If you provide a vector each element will apply to the corresponding columns.
digits.nonzero	The number of digits to keep if the result is close to zero. Sometimes we have an entire table with large numbers only to have a few but interesting observation that are really interesting
txt.NA	The string to exchange NA with
dec	The decimal marker. If the text is in non-English decimal and string formatted you need to change this to the appropriate decimal indicator. The option for this is htmlTable.decimal_marker.
scientific	If the value should be in scientific format.
txtInt_args	A list of arguments to pass to txtInt() if that is to be used for large values that may require a thousands separator. The option for this is htmlTable.round_int.
excl.cols	Columns to exclude from the rounding procedure. This can be either a number or regular expression. Skipped if x is a vector.
excl.rows	Rows to exclude from the rounding procedure. This can be either a number or regular expression.

Value

matrix/data.frame

See Also

Other text formatters: txtInt(), txtMergeLines(), txtPval()

Examples

```
mx <- matrix(c(1, 1.11, 1.25,
               2.50, 2.55, 2.45,
               3.2313, 3, pi),
```

```
             ncol = 3, byrow=TRUE)
txtRound(mx, 1)
```

`vector2string` *Collapse vector to string*

Description

Merges all the values and outputs a string formatted as '1st element', '2nd element', ...

Usage

```
vector2string(
  x,
  quotation_mark = "'",
  collapse = sprintf("%s, %s", quotation_mark, quotation_mark)
)
```

Arguments

`x`	The vector to collapse
`quotation_mark`	The type of quote to use
`collapse`	The string that separates each element

Value

A string with `','` separation

Examples

```
vector2string(1:4)
vector2string(c("a", "b'b", "c"))
vector2string(c("a", "b'b", "c"), quotation_mark = '"')
```

Index