Package ‘htmlTable’

January 7, 2019

Version 1.13.1
Date 2019-01-05
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 http://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, xtable, ztable, Hmisc, reshape, rmarkdown, pander, chron, lubridate, tibble, tidyr (>= 0.7.2), dplyr (>= 0.7.4)

Encoding UTF-8

NeedsCompilation no

VignetteBuilder knitr

RoxygenNote 6.1.1

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

Repository CRAN

Date/Publication 2019-01-07 21:40:12 UTC
**R topics documented:**

concatHtmlTables ........................................... 2
htmlTable ....................................................... 4
htmlTableWidget ............................................. 12
htmlTableWidget-shiny ..................................... 13
interactiveTable ............................................ 13
prConvertDiffFactors ....................................... 15
prEscapeHtml .................................................. 15
SCB ........................................................... 16
tblNoLast ....................................................... 17
tblNoNext ....................................................... 18
tidyHtmlTable ............................................... 18
txtInt .......................................................... 21
txtMergeLines ............................................... 21
txtPval ........................................................ 22
txtRound ....................................................... 23
vector2string ............................................... 24

**Index**

26

---

**concatHtmlTables**

*Function for concatenating htmlTables*

**Description**

Function for concatenating htmlTables

**Usage**

concatHtmlTables(tables, headers)

**Arguments**

- **tables**: A list of html tables 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

# Store all output into a list in order to
# output everything at once at the end
all_tables <- list()

# A simple output
output <- matrix(1:4,
    ncol=2,
    dimnames = list(list("Row 1", "Row 2"),
                    list("Column 1", "Column 2")))
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)
    }
}
htmlTable(output, align="r",
    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;"),
                   c(1,2,NA),
                   c(2,2,2)),
    caption="Basic table with both column spanners (groups) and row groups",
    tfoot="\&dagger; A table footer comment",
    cspan.rgroup = 2,
    col.columns = c(rep("none", 2),
                   rep("\#F5FBFF", 4)),
    col.rgroup = c("none", "\#F7F7F7"),
    css.cell = "padding-left: .5em; padding-right: .2em;"
) ->
    all_tables["Advanced table"]

# An advanced empty table
output <- matrix(ncol = 6,
    nrow = 8)
htmlTable(output, 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 comment",
   cspan.rgroup = 2,
   col.columns = c(rep("none", 2),
   rep("#F5FBBF", 4)),
   col.rgroup = c("none", "#F7F7F7"),
   css.cell = "padding-left: .5em; padding-right: .2em;"") ->
all_tables["Empty table"]

# An example of how to use the css.cell for header styling
simple_output <- matrix(1:4, ncol=2)
htmlTable(simple_output,
   header = LETTERS[1:2],
   css.cell = rbind(rep("background: lightgrey; font-size: 2em;", times=ncol(simple_output)),
   matrix("", ncol=ncol(simple_output), nrow=nrow(simple_output)))) ->
all_tables["Header formatting"]

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

---

**htmlTable**

### Outputting HTML tables

#### Description

This is a function for outputting a more advanced table than what `xtable`, `ztable`, or `knitr`'s `kable()` allows. It's aim is to provide the `Hmisc latex()` colgroup and rowgroup functions in HTML. The html-output is designed for maximum compatibility with LibreOffice/OpenOffice.

#### Usage

```r
htmlTable(x, ...)
```

## Default S3 method:
```r
htmlTable(x, header, rnames, rowlabel, caption, tfoot,
   label, rgroup, n.rgroup, cgroup, n.cgroup, tspanner, n.tspanner, total,
   align = paste(rep("c", ncol(x)), collapse = ""),
   align.header = paste(rep("c", ncol(x)), collapse = ""), align.cgroup,
   css.rgroup = "font-weight: 908;", css.rgroup.sep = "",
```
### S3 method for class 'htmlTable'

`knit_print(x, ...)`

`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.
- `...` Passed on to `print.htmlTable` function and any argument except the `useViewer` will be passed on to the `cat` functions arguments.
- `header` A vector of character strings specifying column header, defaulting to `colnames(x)`
- `rnames` Default rownames are generated from `rownames(x)`. If you provide `FALSE` then it will skip the rownames. *Note:* For data.frames if you do `rownames(my_dataframe) <- NULL` it still has rownames. Thus you need to use `FALSE` if you want to suppress rownames for data.frames.
- `rowlabel` If the table has rownames 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 `<a href="#anchor_name">see table 2</a>`.
  *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 set 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.crgroup
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.crgroup=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.crgroup if all groups have the same number of columns. If the n.crgroup 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".

align
A character strings specifying column alignments, defaulting to `paste(rep('c',ncol(x)),collapse='')` to center. Valid 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 not 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(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

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.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>css.class</strong></td>
<td>The html CSS class for the table. This allows directing html formatting through CSS directly at all instances of that class. <em>Note:</em> unfortunately the CSS is frequently ignored by word processors. This option is mostly intended for web-presentations.</td>
</tr>
<tr>
<td><strong>css.table</strong></td>
<td>You can specify the the style of the table-element using this parameter.</td>
</tr>
<tr>
<td><strong>pos.rowlabel</strong></td>
<td>Where the rowlabel should be positioned. This value can be &quot;top&quot;, &quot;bottom&quot;, &quot;header&quot;, or a integer between 1 and nrow(rgroup) + 1. The options &quot;bottom&quot; and &quot;header&quot; are the same, where the row label is presented at the same level as the header.</td>
</tr>
<tr>
<td><strong>pos.caption</strong></td>
<td>Set to &quot;bottom&quot; to position a caption below the table instead of the default of &quot;top&quot;.</td>
</tr>
<tr>
<td><strong>col.rgroup</strong></td>
<td>Alternating colors (zebra striping/banded rows) for each rgroup; one or two colors is recommended and will be recycled.</td>
</tr>
<tr>
<td><strong>col.columns</strong></td>
<td>Alternating colors for each column.</td>
</tr>
<tr>
<td><strong>padding.rgroup</strong></td>
<td>Generally two non-breakings spaces, i.e.   , but some journals only have a bold face for the rgroup and leaves the subelements unindented.</td>
</tr>
<tr>
<td><strong>padding.tspanner</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>ctable</strong></td>
<td>If the table should have a double top border or a single a’ la LaTeX ctable style compatibility.</td>
</tr>
<tr>
<td><strong>compatibility</strong></td>
<td>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. <em>Note:</em> 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 additionally use the options(htmlTableCompat = &quot;html&quot;) 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 &quot;mso-number-format:&quot;@&quot;.</td>
</tr>
<tr>
<td><strong>cspan.rgroup</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>escape.html</strong></td>
<td>logical: should HTML characters be escaped? Defaults to FALSE.</td>
</tr>
<tr>
<td><strong>useViewer</strong></td>
<td>If you are using RStudio there is a viewer that can render the table within that is envoked if in 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). <em>Note:</em> If you want to force all output to go through the cat() the set options(htmlTable.cat = TRUE).</td>
</tr>
</tbody>
</table>
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 indentation of two blank spaces and are grouped with their corresponing rgroup element. The \texttt{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 \texttt{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 \texttt{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. \texttt{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 \texttt{nrow(x) x ncol(x)} or \texttt{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 \texttt{knitr} outputting \texttt{html}, i.e. markdown mode. As the function returns raw html-code the compatibility with non-html formatting is limited, even with \texttt{pandoc}.

Thanks to the the \texttt{knit_print} and the \texttt{asis_output} the results='asis' is no longer needed except within for-loops. If you have a knit-chunk with a for loop and use \texttt{print()} to produce raw html you must set the chunk option results='asis’. Note: the print-function relies on the \texttt{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 rstudio api and identifies if its a file with and ‘Rmd’ file ending or if ther is an element with ‘html_notebook’). If you don’t want this behaviour 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 = "<b>Table %s:</b> ")` you can manipulate the counter table text that is added prior to the actual caption. Note, you should use the `sprintf` % instead of % 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.

**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 overridden by the align argument.

**Empty dataframes**

An empty dataframe 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.

**Browsers and possible issues**

*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 2013:

- **Internet Explorer (v. 11.20.10586.0)** Works perfectly when copy-pasting into Word
**htmlTable**

- **RStudio** (v. 0.99.448) Works perfectly when copy-pasting into Word. *Note:* can have issues with multiline cgroups - see [bug](#).
- **Chrome** (v. 47.0.2526.106) Works perfectly when copy-pasting into Word. *Note:* can have issues with multiline 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 LibreOffice or Word is no longer recommended. You get much prettier results using the cut-and-paste option.

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, rownames, 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 advized. Most importantly you should try to use `&lt;` instead of `<` and `&gt;` instead of `>`. You can find a complete list of html characters [here](#).

**See Also**

- `txtMergeLines`, `latex`
- Other table functions: `tblNoLast`, `tblNoNext`

**Examples**

```r
# Store all output into a list in order to
# output everything at once at the end
all_tables <- list()

# A simple output
output <- matrix(1:4,
                ncol=2,
                dimnames = list(list("Row 1", "Row 2"),
                                list("Column 1", "Column 2")))

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)
  }
}

htmlTable(output, align="r",
          header = paste(c("1st", "2nd",")
```
```r
htmlTable

"3rd", "4th",
"5th", "6th"),
"hdr"),

rnames = paste(c("1st", "2nd",
"3rd",
paste0(4:8, "th")),
"row"),

ger = paste("Group", LETTERS[1:3]),

ger = c(2,4,nrow(output) - 6),
cg = rbind(c("", "Column spanners", NA),
c("", "Cgroup 1", "Cgroup 2&dagger;")

cg = 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 comment",
cspan.rg = 2,
col.columns = c(rep("none", 2),
rep("#F5FBFF", 4)),
col.rg = c("none", "#F7F7F7"),
css.cell = "padding-left: .5em; padding-right: .2em;"

all_tables[""Advanced table""]

# An advanced empty table

output <- matrix(ncol = 6,
nrow = 0)

htmlTable(output, align="r",

header = paste(c("1st", "2nd",
"3rd", "4th",
"5th", "6th"),
"hdr"),

cg = rbind(c("", "Column spanners", NA),
c("", "Cgroup 1", "Cgroup 2&dagger;")

cg = 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 comment",
cspan.rg = 2,
col.columns = c(rep("none", 2),
rep("#F5FBFF", 4)),
col.rg = c("none", "#F7F7F7"),
css.cell = "padding-left: .5em; padding-right: .2em;"

all_tables[""Empty table""]

# An example of how to use the css.cell for header styling

simple_output <- matrix(1:4, ncol=2)

htmlTable(simple_output,

header = LETTERS[1:2],
css.cell = rbind(rep("background: lightgrey; font-size: 2em;", times=ncol(simple_output)),

matrix("", ncol=ncol(simple_output), nrow=nrow(simple_output))))

all_tables[""Header formatting""]

concatHtmlTables(all_tables)
```
htmlTableWidget  
*htmlTable with pagination widget*

**Description**

This widget renders a table with pagination into an htmlwidget.

**Usage**

```r
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
Description

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

Usage

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

```r
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

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

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

```r
interactiveTable(x, ..., txt.maxlen = 20, button = FALSE,
minimized.columns, js.scripts = c())
```

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

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

```r
## 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 its own `<script>` element.
- `tbl` An htmlTable object can be directly passed into the function
- `useViewer` If you are using RStudio there is a viewer that can render the table within that is envoked if in `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 `cat()` the set `options(htmlTable.cat = TRUE).

Value

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

Examples

```r
# 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...
```
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

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, prAddSemicolon2StrEnd, prGetCgroupHeader,
prGetRowlabelPos, prGetStyle, prPrepareAlign, prPrepareCgroup, prTblNo

---

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

http://scb.se

Examples

```r
## 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(
  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;",
```

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)

tblNoLast

| 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**

```r
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**

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

---

tidyHtmlTable  

*Generate an htmlTable using a ggplot2-like interface*

**Description**

Builds an `htmlTable` by mapping columns from the input data, `x`, to elements of an output `htmlTable` (e.g. rnames, header, etc.)

**Usage**

```r
tidyHtmlTable(x, value = "value", header = "header", rnames = "rnames", rgroup = NULL, hidden_rgroup = NULL, cgroup1 = NULL, cgroup2 = NULL, tspanner = NULL, hidden_tspanner = NULL, ...)
```
### Arguments

- **x**: Tidy data used to build the `htmlTable`
- **value**: The column containing values filling individual cells of the output `htmlTable`
- **header**: The column in `x` specifying column headings
- **rnames**: The column in `x` specifying row names
- **rgroup**: The column in `x` specifying row groups
- **hidden_rgroup**: rgroup values that will be hidden.
- **cgroup1**: The column in `x` specifying the inner most column groups
- **cgroup2**: The column in `x` specifying the outer most column groups
- **tspanner**: The column in `x` specifying tspanner groups
- **hidden_tspanner**: tspanner values that will be hidden.
- ... 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`
- `cgroup1`
- `cgroup2`
- `tspanner`

Note that unlike in `htmlTable` which contains `cgroup`, and which may specify a variable number of column groups, `tidyHtmlTable` contains the parameters `cgroup1` and `cgroup2`. These parameters correspond to the inward most and outward most column groups respectively.

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.
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

Additional dependencies

In order to run this function you also must have dplyr and tidyr packages installed. These have been removed due to the additional 20 Mb that these dependencies added (issue #47). The particular functions required are:

- dplyr: mutate_at, select, pull, slice, filter, arrange_at, mutate_if, is.grouped_df, left_join
- tidyr: spread

See Also

htmlTable

Examples

```r
## Not run:
library(tidyverse)
mtcars %>%
  rownames_to_column %>
  select(rownames, cyl, gear, hp, mpg, qsec) %>
  gather(per_metric, value, 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)) %>
  gather(summary_stat, value, Mean, SD, Min, Max) %>
  ungroup %>
  mutate(gear = paste(gear, "Gears"),
         cyl = paste(cyl, "Cylinders")) %>
  tidyHtmlTable(header = "gear",
                 cgroup1 = "cyl",
                 cell_value = "value",
                 rnames = "summary_stat",
                 rgroup = "per_metric")
```

## End(Not run)
\texttt{txtInt} \hspace{1cm} \textit{SI or English formatting of an integer}

\textbf{Description}

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

\textbf{Usage}

\texttt{txtInt(x, language = "en", html = TRUE, ...)}

\textbf{Arguments}

\begin{itemize}
\item \texttt{x} \hspace{1cm} The integer variable
\item \texttt{language} \hspace{1cm} 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.
\item \texttt{html} \hspace{1cm} If the format is used in html context then the space should be a non-breaking space, \&nbsp;
\item ... \hspace{1cm} Passed to \texttt{format}
\end{itemize}

\textbf{Value}

\texttt{string}

\textbf{Examples}

\begin{verbatim}
txtInt(123)
txtInt(1234)
txtInt(12345)
txtInt(123456)
\end{verbatim}

\texttt{txtMergeLines} \hspace{1cm} \textit{A merges lines while preserving the line break for html/LaTeX}

\textbf{Description}

This function helps you to do a multiline table header in both html and in LaTeX. In html this isn’t that tricky, you just use the \texttt{<br />} command but in LaTeX I often find myself writing vbox/hbox stuff and therefore I’ve created this simple helper function

\textbf{Usage}

\texttt{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 \texttt{br} will be used: \texttt{<br>}, otherwise it uses the \texttt{<br/>} that is compatible with the xhtml-formatting.

Value

string

See Also

Other text formatters: \texttt{txtPval, txtRound}

Examples

\begin{verbatim}
  txtMergeLines("hello", "world")
  txtMergeLines("hello", "world", html=FALSE)
  txtMergeLines("hello", "world", list("A list", "is OK"))
\end{verbatim}

---

\begin{verbatim}
  txtPval(pvalues, lim.2dec = 10^-2, lim.sig = 10^-4, html = TRUE, ...)
\end{verbatim}

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 \textless{} 0.001 as it becomes ridiculous to report anything below that value.

Usage

\begin{verbatim}
  txtPval(pvalues, lim.2dec = 10^-2, lim.sig = 10^-4, html = TRUE, ...)
\end{verbatim}

Arguments

\begin{description}
\item[pvalues] The p-values
\item[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. \textit{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.
\item[lim.sig] The significance limit for the less than sign, i.e. the \texttt{<} sign
\item[html] If the less than sign should be \texttt{<} or \texttt{&lt;} as needed for html output.
\item[... ] Currently only used for generating warnings of deprecated call parameters.
\end{description}
Value

vector

See Also

Other text formatters: txtMergeLines, txtRound

Examples

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

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 = ".", scientific, ...)

## 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, excl.rows, ...)

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
The decimal marker. If the text is in non-english decimal and string formatted you need to change this to the appropriate decimal indicator.

If the value should be in scientific format.

Columns to exclude from the rounding procedure. This can be either a number or regular expression. Skipped if x is a vector.

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: txtMergeLines, txtPval

Examples

```r
mx <- matrix(c(1, 1.1, 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**

```r
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

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

*Topic data
  SCB, 16
asis_output, 8
cat, 5, 7, 14
colnames, 5
concatHtmlTables, 2
dplyr, 20
format, 21
htmlTable, 4, 14, 16–18, 20
htmlTableWidget, 12
htmlTableWidget-shiny, 13
htmlTableWidgetOutput
  (htmlTableWidget-shiny), 13
interactive, 7, 8, 14
interactiveTable, 13
kable, 4
knit_print, 8
knit_print.htmlTable (htmlTable), 4
knit_print.interactiveTable
  (interactiveTable), 13
latex, 4, 10
options, 7, 14
paste, 6
prAddCells, 16
prAddSemicolon2StrEnd, 16
prConvertDfFactors, 15
prEscapeHtml, 15
prGetCgroupHeader, 16
prGetRowLabelPos, 16
prGetStyle, 16
print.htmlTable (htmlTable), 4
print.interactiveTable
  (interactiveTable), 13
prPrepareAlign, 16
prPrepareCgroup, 16
prTblNo, 16
renderHtmlTableWidget
  (htmlTableWidget-shiny), 13
rownames, 5
SCB, 16
sprintf, 9
tblNoLast, 10, 17, 18
tblNoNext, 10, 17, 18
tidyHtmlTable, 18
tidyr, 20
txtInt, 21
txtMergeLines, 5, 10, 21, 23, 24
txtPval, 22, 22, 24
txtRound, 22, 23, 23
vector2string, 24