Package ‘htmltools’

June 16, 2020

Type    Package
Title   Tools for HTML
Version 0.5.0
Description Tools for HTML generation and output.
Depends R (>= 2.14.1)
Imports  utils, digest, grDevices, base64enc, rlang
Suggests markdown, testthat, withr, Cairo, ragg
Enhances knitr
License GPL (>= 2)
URL https://github.com/rstudio/htmltools
BugReports https://github.com/rstudio/htmltools/issues
RoxygenNote 7.1.0.9000
Encoding UTF-8
Collate 'colors.R' 'html_dependency.R' 'html_escape.R' 'html_print.R'
     'images.R' 'known_tags.R' 'shim.R' 'utils.R' 'tags.R'
     'template.R'
NeedsCompilation yes
Author Joe Cheng [aut],
    Carson Sievert [aut, cre] (<https://orcid.org/0000-0002-4958-2844>),
    Winston Chang [aut],
    Yihui Xie [aut],
    Jeff Allen [aut],
    RStudio [cph]
Maintainer Carson Sievert <carson@rstudio.com>
Repository CRAN
Date/Publication 2020-06-16 14:10:10 UTC
### R topics documented:

- as.tags ............................... 2
- browsable ................................ 3
- builder ................................ 4
- capturePlot ........................... 6
- copyDependencyToDir .................. 7
- css ................................ 8
- defaultPngDevice ...................... 9
- findDependencies ...................... 9
- HTML ................................ 10
- htmlDependencies ..................... 10
- htmlDependency ........................ 11
- htmlEscape ......................... 13
- htmlPreserve ......................... 14
- htmlTemplate ......................... 15
- html_print ......................... 16
- include ........................... 16
- knitr_methods ......................... 17
- makeDependencyRelative .............. 18
- parseCssColors ....................... 18
- plotTag ................................ 19
- print.shiny.tag ....................... 21
- renderDependencies .................... 21
- renderDocument ....................... 22
- renderTags ................................ 23
- resolveDependencies .................. 24
- save_html ................................ 24
- singleton ........................... 25
- singleton_tools ....................... 25
- subtractDependencies .................. 26
- suppressDependencies ................ 26
- tag ................................ 27
- urlEncodePath ......................... 28
- validateCssUnit ....................... 29
- withTags ................................ 29

### Index

```
\begin{itemize}
  \item as.tags \chaptermark{as.tags} \indexentry{as.tags}{as.tags (Convert a value to tags)}
\end{itemize}
```

### Description

An S3 method for converting arbitrary values to a value that can be used as the child of a tag or tagList. The default implementation simply calls `as.character`. 

```r
as.tags(value, ...) {
  as.character(value)
}
```
Usage

as.tags(x, ...)

Arguments

x Object to be converted.
... Any additional parameters.

---

browsable Make an HTML object browsable

Description

By default, HTML objects display their HTML markup at the console when printed. browsable can be used to make specific objects render as HTML by default when printed at the console.

Usage

browsable(x, value = TRUE)

is.browsable(x)

Arguments

x The object to make browsable or not.
value Whether the object should be considered browsable.

Details

You can override the default browsability of an HTML object by explicitly passing browse = TRUE (or FALSE) to the print function.

Value

browsable returns x with an extra attribute to indicate that the value is browsable.
is.browsable returns TRUE if the value is browsable, or FALSE if not.
**HTML Builder Functions**

**Description**

Simple functions for constructing HTML documents.

**Usage**

```r
tags
p(..., .noWS = NULL)

h1(..., .noWS = NULL)

h2(..., .noWS = NULL)

h3(..., .noWS = NULL)

h4(..., .noWS = NULL)

h5(..., .noWS = NULL)

h6(..., .noWS = NULL)

a(..., .noWS = NULL)

br(..., .noWS = NULL)

div(..., .noWS = NULL)

span(..., .noWS = NULL)

pre(..., .noWS = NULL)

code(..., .noWS = NULL)

img(..., .noWS = NULL)

strong(..., .noWS = NULL)

em(..., .noWS = NULL)

hr(..., .noWS = NULL)
```
Arguments

... Attributes and children of the element. Named arguments become attributes, and positional arguments become children. Valid children are tags, single-character character vectors (which become text nodes), raw HTML (see HTML), and html_dependency objects. You can also pass lists that contain tags, text nodes, or HTML. To use boolean attributes, use a named argument with a NA value. (see example)

.noWS A character vector used to omit some of the whitespace that would normally be written around this tag. Valid options include before, after, outside, after-begin, before-end, and inside. Any number of these options can be specified.

Details

The tags environment contains convenience functions for all valid HTML5 tags. To generate tags that are not part of the HTML5 specification, you can use the tag() function.

Dedicated functions are available for the most common HTML tags that do not conflict with common R functions.

The result from these functions is a tag object, which can be converted using as.character().

References

• W3C html specification about boolean attributes https://www.w3.org/TR/html5/infrastructure.html#sec-boolean-attributes

Examples

doc <- tags$html(
  tags$head(
    tags$title('My first page')
  ),
  tags$body(
    h1('My first heading'),
    p('My first paragraph, with some ',
      strong('bold'),
      ' text.'),
    div(id='myDiv', class='simpleDiv',
        'Here is a div with some attributes."
    )
  )
)
cat(as.character(doc))

# create an html5 audio tag with controls.
# controls is a boolean attributes
audio_tag <- tags$audio(
  controls = NA,
  tags$source(
    src = "myfile.wav",
    type = "audio/wav"
  )
)
capturePlot

Capture a plot as a saved file

description

Easily generates a .png file (or other graphics file) from a plotting expression.

Usage

capturePlot(
  expr,
  filename = tempfile(fileext = ".png"),
  device = defaultPngDevice(),
  width = 400,
  height = 400,
  res = 72,
  ...
)

Arguments

expr
  A plotting expression that generates a plot (or yields an object that generates a
  plot when printed, like a ggplot2). We evaluate this expression after activating
  the graphics device (device).

filename
  The output filename. By default, a temp file with .png extension will be used;
  you should provide a filename with a different extension if you provide a non-
  PNG graphics device function.

device
  A graphics device function; by default, this will be either grDevices::png() or
  Cairo::CairoPNG(), depending on your system and configuration. See
defaultPngDevice().

width, height, res,...
  Additional arguments to the device function.

See Also

plotTag() saves plots as a self-contained <img> tag.
Examples

# Default settings
res <- capturePlot(plot(cars))
if (interactive()) browseURL(res)

# Use custom width/height
pngpath <- tempfile(fileext = "png")
capturePlot(plot(pressure), pngpath, width = 800, height = 375)
if (interactive()) browseURL(pngpath)

# Use SVG
svgpath <- capturePlot(
  plot(pressure),
  tempfile(fileext = "svg"),
  grDevices::svg,
  width = 8, height = 3.75)
if (interactive()) browseURL(svgpath)

# Clean up
unlink(res)
unlink(pngpath)
unlink(svgpath)

---

**copyDependencyToDir**  
*Copy an HTML dependency to a directory*

### Description
Copies an HTML dependency to a subdirectory of the given directory. The subdirectory name will be *name-version* (for example, "outputDir/jquery-1.11.0"). You may set *options(htmltools.dir.version = FALSE)* to suppress the version number in the subdirectory name.

### Usage
`copyDependencyToDir(dependency, outputDir, mustWork = TRUE)`

### Arguments
- **dependency**: A single HTML dependency object.
- **outputDir**: The directory in which a subdirectory should be created for this dependency.
- **mustWork**: If TRUE and dependency does not point to a directory on disk (but rather a URL location), an error is raised. If FALSE then non-disk dependencies are returned without modification.
Details

In order for disk-based dependencies to work with static HTML files, it's generally necessary to copy them to either the directory of the referencing HTML file, or to a subdirectory of that directory. This function makes it easier to perform that copy.

Value

The dependency with its src value updated to the new location's absolute path.

See Also

makeDependencyRelative can be used with the returned value to make the path relative to a specific directory.

---

css  

CSS string helper

Description

Convenience function for building CSS style declarations (i.e. the string that goes into a style attribute, or the parts that go inside curly braces in a full stylesheet).

Usage

`css(..., collapse_ = "")`

Arguments

...  
Named style properties, where the name is the property name and the argument is the property value. See Details for conversion rules.

collapse_  
(Note that the parameter name has a trailing underscore character.) Character to use to collapse properties into a single string; likely "" (the default) for style attributes, and either "\n" or NULL for style blocks.

Details

CSS uses '-' (minus) as a separator character in property names, but this is an inconvenient character to use in an R function argument name. Instead, you can use '.' (period) and/or '-' (underscore) as separator characters. For example, `css(font.size = "12px")` yields "font-size:12px;".

To mark a property as !important, add a '!' character to the end of the property name. (Since '!' is not normally a character that can be used in an identifier in R, you'll need to put the name in double quotes or backticks.)

Argument values will be converted to strings using `paste(collapse = " ")`. Any property with a value of NULL or "" (after paste) will be dropped.
**Examples**

```r
padding <- 6

css(
    font.family = "Helvetica, sans-serif",
    margin = paste0(c(10, 20, 10, 20), "px"),
    "padding!" = if (!is.null(padding)) padding
)
```

---

**defaultPngDevice**

*Determine the best PNG device for your system*

**Description**

Returns the best PNG-based graphics device for your system, in the opinion of the htmltools maintainers. On Mac, `grDevices::png()` is used; on all other platforms, either `ragg::agg_png()` or `Cairo::CairoPNG()` are used if their packages are installed. Otherwise, `grDevices::png()` is used.

**Usage**

```r
defaultPngDevice()
```

**Value**

A graphics device function.

---

**findDependencies**

*Collect attached dependencies from HTML tag object*

**Description**

Walks a hierarchy of tags looking for attached dependencies.

**Usage**

```r
findDependencies(tags, tagify = TRUE)
```

**Arguments**

- `tags`: A tag-like object to search for dependencies.
- `tagify`: Whether to tagify the input before searching for dependencies.

**Value**

A list of `htmlDependency` objects.
**HTML**  
*Mark Characters as HTML*

---

**Description**

Marks the given text as HTML, which means the `tag` functions will know not to perform HTML escaping on it.

**Usage**

```r
HTML(text, ..., .noWS = NULL)
```

**Arguments**

- `text`: The text value to mark with HTML
- `...`: Any additional values to be converted to character and concatenated together
- `.noWS`: Character vector used to omit some of the whitespace that would normally be written around this HTML. Valid options include `before`, `after`, and `outside` (equivalent to `before` and `end`).

**Value**

The same value, but marked as HTML.

**Examples**

```r
el <- div(HTML("I like <u>turtles</u>"))
cat(as.character(el))
```

---

**htmlDependencies**  
*HTML dependency metadata*

---

**Description**

Gets or sets the HTML dependencies associated with an object (such as a tag).

**Usage**

```r
htmlDependencies(x)

htmlDependencies(x) <- value

attachDependencies(x, value, append = FALSE)
```
htmlDependency

Arguments

- **x**: An object which has (or should have) HTML dependencies.
- **value**: An HTML dependency, or a list of HTML dependencies.
- **append**: If FALSE (the default), replace any existing dependencies. If TRUE, add the new dependencies to the existing ones.

Details

`attachDependencies` provides an alternate syntax for setting dependencies. It is similar to `local({htmlDependencies(x) <- value; x})`, except that if there are any existing dependencies, `attachDependencies` will add to them, instead of replacing them.

As of htmltools 0.3.4, HTML dependencies can be attached without using `attachDependencies`. Instead, they can be added inline, like a child object of a tag or `tagList`.

Examples

```r
# Create a JavaScript dependency
dep <- htmlDependency("jqueryui", "1.11.4", c(href="shared/jqueryui"),
  script = "jquery-ui.min.js")

# A CSS dependency
htmlDependency(
  "font-awesome", "4.5.0", c(href="shared/font-awesome"),
  stylesheet = "css/font-awesome.min.css"
)

# A few different ways to add the dependency to tag objects:
# Inline as a child of the div()
div("Code here", dep)
# Inline in a tagList
tagList(div("Code here"), dep)
# With attachDependencies
attachDependencies(div("Code here"), dep)
```

---

**htmlDependency**  Define an HTML dependency

**Description**

Define an HTML dependency (i.e. CSS and/or JavaScript bundled in a directory). HTML dependencies make it possible to use libraries like jQuery, Bootstrap, and d3 in a more composable and portable way than simply using script, link, and style tags.
**Usage**

```r
textDependencyname, version, src, meta = NULL, script = NULL, stylesheet = NULL, head = NULL, attachment = NULL, package = NULL, all_files = TRUE
)
```

**Arguments**

- `name`: Library name
- `version`: Library version
- `src`: Unnamed single-element character vector indicating the full path of the library directory. Alternatively, a named character string with one or more elements, indicating different places to find the library; see Details.
- `meta`: Named list of meta tags to insert into document head
- `script`: Script(s) to include within the document head (should be specified relative to the `src` parameter).
- `stylesheet`: Stylesheet(s) to include within the document (should be specified relative to the `src` parameter).
- `head`: Arbitrary lines of HTML to insert into the document head
- `attachment`: Attachment(s) to include within the document head. See Details.
- `package`: An R package name to indicate where to find the `src` directory when `src` is a relative path (see `resolveDependencies`).
- `all_files`: Whether all files under the `src` directory are dependency files. If FALSE, only the files specified in `script`, `stylesheet`, and `attachment` are treated as dependency files.

**Details**

Each dependency can be located on the filesystem, at a relative or absolute URL, or both. The location types are indicated using the names of the `src` character vector: `file` for filesystem directory, `href` for URL. For example, a dependency that was both on disk and at a URL might use `src = c(file=filepath, href=url)`.

`attachment` can be used to make the indicated files available to the JavaScript on the page via URL. For each element of `attachment`, an element `<link id="DEPNAME-ATTACHINDEX-attachment" rel="attachment" href="...">` is inserted, where `DEPNAME` is name. The value of `ATTACHINDEX` depends on whether `attachment` is named or not; if so, then it’s the name of the element, and if not, it’s the 1-based index of the element. JavaScript can retrieve the URL using something...
like `document.getElementById(depname + "-" + index + "-attachment").href`. Note that depending on the rendering context, the runtime value of the href may be an absolute, relative, or data URI.

`htmlDependency` should not be called from the top-level of a package namespace with absolute paths (or with paths generated by `system.file()`) and have the result stored in a variable. This is because, when a binary package is built, R will run `htmlDependency` and store the path from the building machine’s in the package. This path is likely to differ from the correct path on a machine that downloads and installs the binary package. If there are any absolute paths, instead of calling `htmlDependency` at build-time, it should be called at run-time. This can be done by wrapping the `htmlDependency` call in a function.

**Value**

An object that can be included in a list of dependencies passed to `attachDependencies`.

**See Also**

Use `attachDependencies` to associate a list of dependencies with the HTML it belongs with.

---

### htmlEscape

**Escape HTML entities**

**Description**

Escape HTML entities contained in a character vector so that it can be safely included as text or an attribute value within an HTML document

**Usage**

`htmlEscape(text, attribute = FALSE)`

**Arguments**

- `text`: Text to escape
- `attribute`: Escape for use as an attribute value

**Value**

Character vector with escaped text.
**htmlPreserve**

*Preserve HTML regions*

**Description**

Use "magic" HTML comments to protect regions of HTML from being modified by text processing tools.

**Usage**

```r
htmlPreserve(x)
extractPreserveChunks(strval)
restorePreserveChunks(strval, chunks)
```

**Arguments**

- `x`: A character vector of HTML to be preserved.
- `strval`: Input string from which to extract/restore chunks.
- `chunks`: The `chunks` element of the return value of `extractPreserveChunks`.

**Details**

Text processing tools like markdown and pandoc are designed to turn human-friendly markup into common output formats like HTML. This works well for most prose, but components that generate their own HTML may break if their markup is interpreted as the input language. The `htmlPreserve` function is used to mark regions of an input document as containing pure HTML that must not be modified. This is achieved by substituting each such region with a benign but unique string before processing, and undoing those substitutions after processing.

**Value**

- `htmlPreserve` returns a single-element character vector with "magic" HTML comments surrounding the original text (unless the original text was empty, in which case an empty string is returned).
- `extractPreserveChunks` returns a list with two named elements: `value` is the string with the regions replaced, and `chunks` is a named character vector where the names are the IDs and the values are the regions that were extracted.
- `restorePreserveChunks` returns a character vector with the chunk IDs replaced with their original values.

**Examples**

```
# htmlPreserve will prevent "<script>alert(10*2*3);</script>"
# from getting an <em> tag inserted in the middle
markup <- paste(sep = "\n",
    "This is *emphasized* text in markdown.",
)
htmlPreserve("<script>alert(10*2*3);</script>", "Here is some more *emphasized text*.")
extracted <- extractPreserveChunks(markup)
markup <- extracted$value
# Just think of this next line as Markdown processing
output <- gsub("\*(.*)\*\", "<em>\1</em>", markup)
output <- restorePreserveChunks(output, extracted$chunks)
output

---

**htmlTemplate**

*Process an HTML template*

**Description**

Process an HTML template and return a tagList object. If the template is a complete HTML document, then the returned object will also have class `html_document`, and can be passed to the function `renderDocument` to get the final HTML text.

**Usage**

```r
htmlTemplate(filename = NULL, ..., text_ = NULL, document_ = "auto")
```

**Arguments**

- `filename` Path to an HTML template file. Incompatible with `text_`
- `...` Variable values to use when processing the template.
- `text_` A string to use as the template, instead of a file. Incompatible with `filename`.
- `document_` Is this template a complete HTML document (`TRUE`), or a fragment of HTML that is to be inserted into an HTML document (`FALSE`)? With "auto" (the default), auto-detect by searching for the string "<HTML>" within the template.

**See Also**

`renderDocument`
html_print  

**Implementation of the print method for HTML**

**Description**

Convenience method that provides an implementation of the print method for HTML content.

**Usage**

```r
html_print(
  html,
  background = "white",
  viewer = getOption("viewer", utils::browseURL)
)
```

**Arguments**

- **html**: HTML content to print
- **background**: Background color for web page
- **viewer**: A function to be called with the URL or path to the generated HTML page. Can be NULL, in which case no viewer will be invoked.

**Value**

Invisibly returns the URL or path of the generated HTML page.

---

include  

**Include Content From a File**

**Description**

Load HTML, text, or rendered Markdown from a file and turn into HTML.

**Usage**

```r
includeHTML(path)
includeText(path)
includeMarkdown(path)
includeCSS(path, ...)
includeScript(path, ...)
```
Arguments

path  The path of the file to be included. It is highly recommended to use a relative path (the base path being the Shiny application directory), not an absolute path.
...  Any additional attributes to be applied to the generated tag.

Details

These functions provide a convenient way to include an extensive amount of HTML, textual, Markdown, CSS, or JavaScript content, rather than using a large literal R string.

Note

includeText escapes its contents, but does no other processing. This means that hard breaks and multiple spaces will be rendered as they usually are in HTML: as a single space character. If you are looking for preformatted text, wrap the call with pre, or consider using includeMarkdown instead.

The includeMarkdown function requires the markdown package.

---

knitr_methods  Knitr S3 methods

Description

These S3 methods are necessary to allow HTML tags to print themselves in knitr/markdown documents.

Usage

knit_print.shiny.tag(x, ...)

knit_print.html(x, ...)

knit_print.shiny.tag.list(x, ...)

Arguments

x  Object to knit_print

...  Additional knit_print arguments
**makeDependencyRelative**

*Make an absolute dependency relative*

**Description**

Change a dependency’s absolute path to be relative to one of its parent directories.

**Usage**

```r
makeDependencyRelative(dependency, basepath, mustWork = TRUE)
```

**Arguments**

- `dependency`: A single HTML dependency with an absolute path.
- `basepath`: The path to the directory that `dependency` should be made relative to.
- `mustWork`: If `TRUE` and `dependency` does not point to a directory on disk (but rather a URL location), an error is raised. If `FALSE` then non-disk dependencies are returned without modification.

**Value**

The dependency with its `src` value updated to the new location’s relative path.

If `basepath` did not appear to be a parent directory of the dependency’s directory, an error is raised (regardless of the value of `mustWork`).

**See Also**

- `copyDependencyToDir`

---

**parseCssColors**

*Parse CSS color strings*

**Description**

Parses/normalizes CSS color strings, and returns them as strings in “#RRGGBB” and/or “#RRGGBBAA” format. Understands hex colors in 3, 4, 6, and 8 digit forms, `rgb()`/`rgba()`, `hsl()`/`hsla()`, and color keywords.

**Usage**

```r
parseCssColors(str, mustWork = TRUE)
```
Arguments

str CSS color strings
mustWork If true, invalid color strings will cause an error; if false, then the result will contain NA for invalid colors.

Details

Note that parseCssColors may return colors in #RRGGBBAA format. Such values are not understood by Internet Explorer, and must be converted to rgba(red,green,blue,alpha) format to be safe for the web.

Value

A vector of strings in #RRGGBB or #RRGGBBAA format (the latter is only used for colors whose alpha values are less than FF), or NA for invalid colors when mustWork is false. Such strings are suitable for use in plots, or parsing with col2rgb() (be sure to pass alpha = TRUE to prevent the alpha channel from being discarded).

Examples

parseCssColors(c(
    "#0d6efd",
    "#DC35457F",
    "rgb(32,201,151)",
    "rgba(23,162,184,0.5)",
    "hsl(261,51%,51%)",
    "cornflowerblue"
))

plotTag

Capture a plot as a self-contained <img> tag

Description

Capture a plot as a self-contained <img> tag

Usage

plotTag(
    expr, 
    alt, 
    device = defaultPngDevice(), 
    width = 400, 
    height = 400, 
    pixelratio = 2, 
    mimeType = "image/png", 
)
Arguments

expr  A plotting expression that generates a plot (or yields an object that generates a plot when printed, like a ggplot2).

alt  A single-element character vector that contains a text description of the image. This is used by accessibility tools, such as screen readers for vision impaired users.

device  A graphics device function; by default, this will be either \texttt{\textbf{grDevices::png()}}. \texttt{\textbf{ragg::agg_png()}}. or \texttt{\textbf{Cairo::CairoPNG()}}. depending on your system and configuration. See \texttt{\textbf{defaultPngDevice()}}.

width, height  The width/height that the generated tag should be displayed at, in logical (browser) pixels.

pixelratio  Indicates the ratio between physical and logical units of length. For PNGs that may be displayed on high-DPI screens, use 2; for graphics devices that express width/height in inches (like \texttt{\textbf{grDevices::svg()}}), try 1/72 or 1/96.

mimeType  The MIME type associated with the \texttt{\textbf{device}}. Examples are \texttt{\textbf{image/png}}, \texttt{\textbf{image/tiff}}, \texttt{\textbf{image/svg+xml}}.

deviceArgs  A list of additional arguments that should be included when the \texttt{\textbf{device}} function is invoked.

attribs  A list of additional attributes that should be included on the generated \texttt{\textbf{<img}}} (e.g. \texttt{\textbf{id}}, \texttt{\textbf{class}}).

suppressSize  By default, \texttt{\textbf{plotTag}} will include a style attribute with width and height properties specified in pixels. If you’d rather specify the image size using other methods (like responsive CSS rules) you can use this argument to suppress width ("x"), height ("y"), or both ("xy") properties.

Value

A \texttt{\textbf{browsable()}} HTML \texttt{\textbf{<img}}} tag object. Print it at the console to preview, or call \texttt{\textbf{as.character()}} on it to view the HTML source.

See Also

\texttt{\textbf{capturePlot()}} saves plots as an image file.

Examples

\begin{verbatim}
img <- plotTag(
plot(cars)
), "A plot of the 'cars' dataset", width = 375, height = 275)
\end{verbatim}
if (interactive()) img

svg <- plotTag(plot(pressure), "A plot of the 'pressure' dataset",
device = grDevices::svg, width = 375, height = 275, pixelratio = 1/72,
mimeType = "image/svg+xml")

if (interactive()) svg

print.shiny.tag  

Print method for HTML/tags

Description
S3 method for printing HTML that prints markup or renders HTML in a web browser.

Usage
## S3 method for class 'shiny.tag'
print(x, browse = is.browsable(x), ...)

## S3 method for class 'html'
print(x, ..., browse = is.browsable(x))

Arguments

x
The value to print.

browse
If TRUE, the HTML will be rendered and displayed in a browser (or possibly another HTML viewer supplied by the environment via the viewer option). If FALSE then the HTML object’s markup will be rendered at the console.

...
Additional arguments passed to print.

renderDependencies  

Create HTML for dependencies

Description

Create the appropriate HTML markup for including dependencies in an HTML document.

Usage

renderDependencies(
dependencies,
srcType = c("href", "file"),
encodeFunc = urlEncodePath,
hrefFilter = identity
)
renderDocument

Arguments

dependencies A list of htmlDependency objects.
srcType The type of src paths to use; valid values are file or href.
encodeFunc The function to use to encode the path part of a URL. The default should generally be used.
hrefFilter A function used to transform the final, encoded URLs of script and stylesheet files. The default should generally be used.

Value

An HTML object suitable for inclusion in the head of an HTML document.

Description

This function renders html_document objects, and returns a string with the final HTML content. It calls the renderTags function to convert any shiny.tag objects to HTML. It also finds any web dependencies (created by htmlDependency) that are attached to the tags, and inserts those. To do the insertion, this function finds the string "<!--HEAD_CONTENT -->" in the document, and replaces it with the web dependencies.

Usage

renderDocument(x, deps = NULL, processDep = identity)

Arguments

x An object of class html_document, typically generated by the htmlTemplate function.
deps Any extra web dependencies to add to the html document. This can be an object created by htmlDependency, or a list of such objects. These dependencies will be added first, before other dependencies.
processDep A function that takes a "raw" html_dependency object and does further processing on it. For example, when renderDocument is called from Shiny, the function createWebDependency is used; it modifies the href and tells Shiny to serve a particular path on the filesystem.
renderTags  

**Description**

Renders tags (and objects that can be converted into tags using `as.tags`) into HTML. (Generally intended to be called from web framework libraries, not directly by most users–see `print.html(browse=TRUE)` for higher level rendering.)

**Usage**

```r
renderTags(x, singletons = character(0), indent = 0)
doRenderTags(x, indent = 0)
```

**Arguments**

- `x` Tag object(s) to render
- `singletons` A list of singleton signatures to consider already rendered; any matching singletons will be dropped instead of rendered. (This is useful (only?) for incremental rendering.)
- `indent` Initial indent level, or `FALSE` if no indentation should be used.

**Details**

doRenderTags is intended for very low-level use; it ignores singleton, head, and dependency handling, and simply renders the given tag objects as HTML.

**Value**

renderTags returns a list with the following variables:

- `head` An HTML string that should be included in `<head>`.
- `singletons` Character vector of singleton signatures that are known after rendering.
- `dependencies` A list of resolved `htmlDependency` objects.
- `html` An HTML string that represents the main HTML that was rendered.

doRenderTags returns a simple HTML string.
**save_html**

**Description**

Save the specified HTML object to a file, copying all of it’s dependencies to the directory specified via libdir.

**Usage**

```r
save_html(html, file, background = "white", libdir = "lib")
```

**Arguments**

- `html`: HTML content to print
- `file`: File to write content to
- `background`: Background color for web page
- `libdir`: Directory to copy dependencies to

---

**resolveDependencies**

**Description**

Resolve a list of dependencies

Given a list of dependencies, removes any redundant dependencies (based on name equality). If multiple versions of a dependency are found, the copy with the latest version number is used.

**Usage**

```r
resolveDependencies(dependencies, resolvePackageDir = TRUE)
```

**Arguments**

- `dependencies`: A list of `htmlDependency` objects.
- `resolvePackageDir`: Whether to resolve the relative path to an absolute path via `system.file` when the package attribute is present in a dependency object.

**Value**

- `dependencies`: A list of `htmlDependency` objects with redundancies removed.
singleton

Description

Use `singleton` to wrap contents (tag, text, HTML, or lists) that should be included in the generated document only once, yet may appear in the document-generating code more than once. Only the first appearance of the content (in document order) will be used.

Usage

`singleton(x, value = TRUE)`

`is.singleton(x)`

Arguments

- `x` A tag, text, HTML, or list.
- `value` Whether the object should be a singleton.

singleton_tools

Singleton manipulation functions

Description

Functions for manipulating `singleton` objects in tag hierarchies. Intended for framework authors.

Usage

`surroundSingletons(ui)`

`takeSingletons(ui, singletons = character(0), desingleton = TRUE)`

Arguments

- `ui` Tag object or lists of tag objects. See `builder` topic.
- `singletons` Character vector of singleton signatures that have already been encountered (i.e. returned from previous calls to `takeSingletons`).
- `desingleton` Logical value indicating whether singletons that are encountered should have the singleton attribute removed.
suppressDependencies

Value

surroundSingletons pre-processes a tag object by changing any singleton X into <!–SHINY.SINGLETON[sig]–> X’ <!–/SHINY.SINGLETON[sig]–> where sig is the sha1 of X, and X’ is X minus the singleton attribute.

takeSingletons returns a list with the elements ui (the processed tag objects with any duplicate singleton objects removed) and singletons (the list of known singleton signatures).

subtractDependencies  Subtract dependencies

Description

Remove a set of dependencies from another list of dependencies. The set of dependencies to remove can be expressed as either a character vector or a list; if the latter, a warning can be emitted if the version of the dependency being removed is later than the version of the dependency object that is causing the removal.

Usage

subtractDependencies(dependencies, remove, warnOnConflict = TRUE)

Arguments

dependencies  A list of htmlDependency objects from which dependencies should be removed.

remove  A list of htmlDependency objects indicating which dependencies should be removed, or a character vector indicating dependency names.

warnOnConflict  If TRUE, a warning is emitted for each dependency that is removed if the corresponding dependency in remove has a lower version number. Has no effect if remove is provided as a character vector.

Value

A list of htmlDependency objects that don’t intersect with remove.

suppressDependencies  Suppress web dependencies

Description

This suppresses one or more web dependencies. It is meant to be used when a dependency (like a JavaScript or CSS file) is declared in raw HTML, in an HTML template.

Usage

suppressDependencies(...)

Arguments

... Names of the dependencies to suppress. For example, "jquery" or "bootstrap".

See Also

htmlTemplate for more information about using HTML templates.
htmlDependency

---

tag 

HTML Tag Object

Description

tag() creates an HTML tag definition. Note that all of the valid HTML5 tags are already defined in the tags environment so these functions should only be used to generate additional tags. tag AppendChild() and tag List() are for supporting package authors who wish to create their own sets of tags; see the contents of bootstrap.R for examples.

Usage

tagList(...)
tagAppendAttributes(tag, ...)
tagHasAttribute(tag, attr)
tagGetAttribute(tag, attr)
tagAppendChild(tag, child)
tagAppendChildren(tag, ..., list = NULL)
tagSetChildren(tag, ..., list = NULL)
tag("_tag_name", varArgs, .noWS = NULL)

Arguments

... Unnamed items that comprise this list of tags.
tag A tag to append child elements to.
attr The name of an attribute.
child A child element to append to a parent tag.
list An optional list of elements. Can be used with or instead of the ... items.
_tag_name HTML tag name
urlEncodePath

varArgs
List of attributes and children of the element. Named list items become attributes, and unnamed list items become children. Valid children are tags, single-character character vectors (which become text nodes), and raw HTML (see HTML). You can also pass lists that contain tags, text nodes, and HTML.

.noWS
Character vector used to omit some of the whitespace that would normally be written around this tag. Valid options include before, after, outside, after-begin, and before-end. Any number of these options can be specified.

Value
An HTML tag object that can be rendered as HTML using as.character().

Examples

tagList(tags$h1("Title"),
        tags$h2("Header text"),
        tags$p("Text here"))

    # Can also convert a regular list to a tagList (internal data structure isn't
    # exactly the same, but when rendered to HTML, the output is the same).
    x <- list(tags$h1("Title"),
              tags$h2("Header text"),
              tags$p("Text here"))
    tagList(x)

    # suppress the whitespace between tags
    oneline <- tag("span",
                   tag("strong", "Super strong", .noWS="outside")
    )
    cat(as.character(oneline))

urlEncodePath

Encode a URL path

Description
Encode characters in a URL path. This is the same as URLencode with reserved = TRUE except that / is preserved.

Usage
urlEncodePath(x)

Arguments

  x A character vector.
validateCssUnit

Validate proper CSS formatting of a unit

**Description**

Checks that the argument is valid for use as a CSS unit of length.

**Usage**

```r
validateCssUnit(x)
```

**Arguments**

- `x` The unit to validate. Will be treated as a number of pixels if a unit is not specified.

**Details**

- NULL and NA are returned unchanged.
- Single element numeric vectors are returned as a character vector with the number plus a suffix of "px".
- Single element character vectors must be "auto" or "inherit", a number, or a length calculated by the "calc" CSS function. If the number has a suffix, it must be valid: px, %, ch, em, rem, pt, in, cm, mm, ex, pc, vh, vw, vmin, or vmax. If the number has no suffix, the suffix "px" is appended.
- Any other value will cause an error to be thrown.

**Value**

A properly formatted CSS unit of length, if possible. Otherwise, will throw an error.

**Examples**

```r
validateCssUnit("10%")
validateCssUnit(400)  # treated as '400px'
```

withTags

Evaluate an expression using tags

**Description**

This function makes it simpler to write HTML-generating code. Instead of needing to specify tags each time a tag function is used, as in `tags$div()` and `tags$p()`, code inside withTags is evaluated with tags searched first, so you can simply use `div()` and `p()`.
Usage

withTags(code)

Arguments

code A set of tags.

Details

If your code uses an object which happens to have the same name as an HTML tag function, such as source() or summary(), it will call the tag function. To call the intended (non-tags function), specify the namespace, as in base::source() or base::summary().

Examples

# Using tags$ each time
tags$div(class = "myclass", 
tags$h3("header"), 
tags$p("text")
)

# Equivalent to above, but using withTags
withTags(
  div(class = "myclass", 
  h3("header"), 
  p("text")
)
)
Index

a (builder), 4
as.character, 2, 5, 28
as.character(), 20
as.tags, 2, 23
attachDependencies, 13
attachDependencies (htmlDependencies), 10
br (builder), 4
browsable, 3
browsable(), 20
builder, 4, 25
Cairo::CairoPNG(), 6, 9, 20
capturePlot, 6
capturePlot(), 20
code (builder), 4
col2rgb(), 19
copyDependencyToDir, 7, 18
createWebDependency, 22
css, 8
defaultPngDevice, 9
defaultPngDevice(), 6, 20
div (builder), 4
doRenderTags (renderTags), 23
em (builder), 4
extractPreserveChunks (htmlPreserve), 14
findDependencies, 9
grDevices::png(), 6, 9, 20
grDevices::svg(), 20
h1 (builder), 4
h2 (builder), 4
h3 (builder), 4
h4 (builder), 4
h5 (builder), 4
h6 (builder), 4
hr (builder), 4
HTML, 5, 10, 22, 23, 25, 28
html_print, 16
htmlDependencies, 10
htmlDependencies<-(htmlDependencies), 10
htmlDependency, 9, 11, 22–24, 26, 27
htmlEscape, 13
htmlPreserve, 14
htmlTemplate, 15, 22, 27
img (builder), 4
include, 16
includeCSS (include), 16
includeHTML (include), 16
includeMarkdown (include), 16
includeScript (include), 16
includeText (include), 16
is.browsable (browsable), 3
is.singleton (singleton), 25
knit_print.html (knitr_methods), 17
knit_print.shiny.tag (knitr_methods), 17
knitr_methods, 17
makeDependencyRelative, 8, 18
p (builder), 4
parseCssColors, 18
plotTag, 19
plotTag(), 6
pre, 17
pre (builder), 4
print, 16
print.html, 23
print.html (print.shiny.tag), 21
print.shiny.tag, 21
ragg::agg_png(), 6, 9, 20
renderDependencies, 21
renderDocument, 15, 22
renderTags, 22, 23
resolved, 23
resolveDependencies, 12, 24
restorePreserveChunks (htmlPreserve), 14

save_html, 24
singleton, 23, 25, 25
singleton_tools, 25
span (builder), 4
strong (builder), 4
subtractDependencies, 26
suppressDependencies, 26
surroundSingletons (singleton_tools), 25
system.file, 24

tag, 5, 10, 25, 27
tagAppendAttributes (tag), 27
tagAppendChild (tag), 27
tagAppendChildren (tag), 27
tagGetAttribute (tag), 27
tagHasAttribute (tag), 27
tagList, 11
tagList (tag), 27
tags, 27
tags (builder), 4
tagSetChildren (tag), 27
takeSingletons (singleton_tools), 25

URLencode, 28
urlEncodePath, 28

validateCssUnit, 29

withTags, 29