Package ‘htmltools’

March 26, 2024

Type Package
Title Tools for HTML
Version 0.5.8
Description Tools for HTML generation and output.
License GPL (>= 2)
URL https://github.com/rstudio/htmltools,
https://rstudio.github.io/htmltools/
BugReports https://github.com/rstudio/htmltools/issues
Depends R (>= 2.14.1)
Imports base64enc, digest, fastmap (>= 1.1.0), grDevices, rlang (>= 1.0.0), utils
Suggests Cairo, markdown, ragg, shiny, testthat, withr
Enhances knitr
Config/Needs/check knitr
Config/Needs/website rstudio/quillt, bench
Encoding UTF-8
RoxygenNote 7.3.1
 ‘template.R’
NeedsCompilation yes
Author Joe Cheng [aut],
Carson Sievert [aut, cre] (https://orcid.org/0000-0002-4958-2844),
Barret Schloerke [aut] (https://orcid.org/0000-0001-9986-114X),
Winston Chang [aut] (https://orcid.org/0000-0002-1576-2126),
Yihui Xie [aut],
Jeff Allen [aut],
Posit Software, PBC [cph, fnd]
Maintainer Carson Sievert <carson@posit.co>
R topics documented:

as.tags .............................................. 3
bindFillRole ........................................ 3
browsable .......................................... 4
builder .............................................. 5
capturePlot ........................................ 7
copyDependencyToDir ............................. 9
css .................................................. 10
defaultPngDevice ................................. 11
findDependencies .................................. 11
HTML ............................................... 12
htmlDependencies ................................. 12
htmlDependency ................................. 13
htmlEscape ........................................ 15
htmlPreserve ..................................... 16
htmlTemplate .................................... 17
html_print ......................................... 18
include ............................................ 18
knitr_methods ..................................... 19
makeDependencyRelative ....................... 20
parseCssColors .................................... 20
plotTag ............................................ 21
print.shiny.tag ................................. 23
renderDependencies ............................. 23
renderDocument .................................. 24
renderTags ....................................... 25
resolveDependencies ......................... 26
save_html ......................................... 26
singleton ......................................... 27
singleton_tools .................................. 27
subtractDependencies ....................... 28
suppressDependencies ....................... 28
tagAddRenderHook ............................... 29
tagAppendAttributes ......................... 30
tagAppendChild ................................ 31
tagFunction ...................................... 32
tagList ........................................... 32
tagQuery .......................................... 34
urlEncodePath .................................... 36
validateCssUnit .................................. 37
withTags ......................................... 38

Index 40
as.tags

**Convert a value to tags**

**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()`.

**Usage**

```r
as.tags(x, ...)"'
```

**Arguments**

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

---

bindFillRole

**Allow tags to intelligently fill their container**

**Description**

Create fill containers and items. If a fill item is a direct child of a fill container, and that container has an opinionated height, then the item is allowed to grow and shrink to its container’s size.

**Usage**

```r
bindFillRole(
  x, 
  ..., 
  item = FALSE, 
  container = FALSE, 
  overwrite = FALSE, 
  .cssSelector = NULL
)
```

**Arguments**

- `x` a `tag()` object. Can also be a valid `tagQuery()` input if `.cssSelector` is specified.
- `...` currently unused.
- `item` whether or not to treat `x` as a fill item.
container whether or not to treat \( x \) as a fill container. Note, this will set the CSS display property on the tag to flex which can change how its direct children are rendered. Thus, one should be careful not to mark a tag as a fill container when it needs to rely on other display behavior.

overwrite whether or not to override previous calls to `bindFillRole()` (e.g., to remove the item/container role from a tag).

\( .\text{cssSelector} \) A character string containing a CSS selector for targeting particular (inner) tag(s) of interest. For more details on what selector(s) are supported, see `tagAppendAttributes()`.

Value

The original tag object (\( x \)) with additional attributes (and a `htmlDependency()`).

Examples

tagz <- div(
  id = "outer",
  style = css(
    height = "600px",
    border = "3px red solid"
  ),
  div(
    id = "inner",
    style = css(
      height = "400px",
      border = "3px blue solid"
    )
  )
)

# Inner doesn't fill outer
if (interactive()) browsable(tagz)

tagz <- bindFillRole(tagz, container = TRUE)
tagz <- bindFillRole(tagz, item = TRUE, .cssSelector = "#inner")

# Inner does fill outer
if (interactive()) browsable(tagz)

---

**browsable**  

\( \text{Make an HTML object browsable} \)

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.

Description

Create an R object that represents an HTML tag. For convenience, common HTML tags (e.g., <div>) can be created by calling for their tag name directly (e.g., div()). To create less common HTML5 (or SVG) tags (e.g., <article>), use the tags list collection (e.g., tags$article()). To create other non HTML/SVG tags, use the lower-level tag() constructor.

Usage

tags

p(..., .noWS = NULL, .renderHook = NULL)
h1(..., .noWS = NULL, .renderHook = NULL)
h2(..., .noWS = NULL, .renderHook = NULL)
h3(..., .noWS = NULL, .renderHook = NULL)
h4(..., .noWS = NULL, .renderHook = NULL)
h5(..., .noWS = NULL, .renderHook = NULL)
h6(..., .noWS = NULL, .renderHook = NULL)
a(..., .noWS = NULL, .renderHook = NULL)
br(..., .noWS = NULL, .renderHook = NULL)
div(..., .noWS = NULL, .renderHook = NULL)
span(..., .noWS = NULL, .renderHook = NULL)
pre(..., .noWS = NULL, .renderHook = NULL)
code(..., .noWS = NULL, .renderHook = NULL)
img(..., .noWS = NULL, .renderHook = NULL)
strong(..., .noWS = NULL, .renderHook = NULL)
em(..., .noWS = NULL, .renderHook = NULL)
hr(..., .noWS = NULL, .renderHook = NULL)
tag(`_tag_name`, varArgs, .noWS = NULL, .renderHook = NULL)

Arguments

... Tag attributes (named arguments) and children (unnamed arguments). A named argument with an NA value is rendered as a boolean attributes (see example). Children may include any combination of:
- Other tags objects
- HTML() strings
- htmlDependency()s
- Single-element atomic vectors
- list()s containing any combination of the above

.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.

.renderHook A function (or list of functions) to call when the tag is rendered. This function should have at least one argument (the tag) and return anything that can be converted into tags via as.tags(). Additional hooks may also be added to a particular tag via tagAddRenderHook().

._tag_name A character string to use for the tag name.

.varArgs List of tag attributes and children.

Value

A list() with a shiny.tag class that can be converted into an HTML string via as.character() and saved to a file with save_html().
**See Also**

tagList().withTags().tagAppendAttributes().tagQuery()

**Examples**

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.'
    )
  )
)

# html5 <audio> with boolean control attribute
# https://www.w3.org/TR/html5/infrastructure.html#sec-boolean-attributes
tags$audio(
  controls = NA,
  tags$source(
    src = "myfile.wav",
    type = "audio/wav"
  )
)

# suppress the whitespace between tags
tags$span(
  tags$strong("I'm strong", .noWS="outside")
)

---

capturePlot  
*Capture a plot as a saved file*

capturePlot  

**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(), ragg::agg_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))

# View result
if (interactive()) browseURL(res)

# Clean up
unlink(res)

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

# Use a custom graphics device (e.g., SVG)
if (capabilities("cairo")) {
  svgsupath <- capturePlot(
    plot(pressure),
    tempfile(fileext = ".svg"),
    grDevices::svg,
    width = 8, height = 3.75
  )
  if (interactive()) browseURL(svgsupath)
  unlink(svgsupath)
}

copyDependencyToDir

---

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

---

**Description**

Copy 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 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

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 input `text`, 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)
hhtmlDependencies(x) <- value
attachDependencies(x, value, append = FALSE)
```
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.

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

Usage

htmlDependency(
  name,
  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 depen-
dency files.

Details

Each dependency can be located on the filesystem, at a relative or absolute URL, or both. The loca-
tion 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).

script can be given as one of the following:

- a character vector specifying various scripts to include relative to the value of src. Each is
  expanded into its own <script> tag
- A named list with any of the following fields:
- `src`,
- `integrity`, &
- `crossorigin`,
- any other valid `<script>` attributes.

allowing the use of SRI to ensure the integrity of packages downloaded from remote servers. Eg: `script = list(src = "min.js", integrity = "hash")`

- An unnamed list, containing a combination of named list with the fields mentioned previously, and strings. Eg:
  - `script = list(list(src = "min.js"), "util.js", list(src = "log.js"))`
  - `script = "pkg.js"` is equivalent to
  - `script = list(src = "pkg.js")`.

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"`). 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. The shape of the `htmlDependency` object is described (in TypeScript code) here.

---

### 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Text to escape</td>
</tr>
<tr>
<td>attribute</td>
<td>Escape for use as an attribute value</td>
</tr>
</tbody>
</table>

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

htmlPreserve(x)

extractPreserveChunks(strval)

restorePreserveChunks(strval, chunks)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>A character vector of HTML to be preserved.</td>
</tr>
<tr>
<td>strval</td>
<td>Input string from which to extract/restore chunks.</td>
</tr>
<tr>
<td>chunks</td>
<td>The chunks element of the return value of \texttt{extractPreserveChunks}.</td>
</tr>
</tbody>
</table>

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 \texttt{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 `base::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, ..., inline = FALSE)
knit_print.html(x, ..., inline = FALSE)
knit_print.shiny.tag.list(x, ..., inline = FALSE)
knit_print.html_dependency(x, ..., inline = FALSE)
```

**Arguments**

- **x** Object to `knit_print`
- ... Additional `knit_print` arguments
- **inline** Whether or not the code chunk is inline.
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 #RRGBBAA 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 #RRGBB or #RRGBBAA 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",  
  deviceArgs = list(),  
)
plotTag

```r
attrs = list(),
suppressSize = c("none", "x", "y", "xy")
)
```

**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 `grDevices::png()`, `ragg::agg_png()`, or `Cairo::CairoPNG()`, depending on your system and configuration. See `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 `grDevices::svg()`), try 1/72 or 1/96.

- **mimeType**
  The MIME type associated with the device. Examples are `image/png`, `image/tiff`, `image/svg+xml`.

- **deviceArgs**
  A list of additional arguments that should be included when the device function is invoked.

- **attrs**
  A list of additional attributes that should be included on the generated `<img>` (e.g. `id`, `class`).

- **suppressSize**
  By default, `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 `browsable()` HTML `<img>` tag object. Print it at the console to preview, or call `as.character()` on it to view the HTML source.

**See Also**

- `capturePlot()` saves plots as an image file.

**Examples**

```r
img <- plotTag({
  plot(cars)
}, "A plot of the 'cars' dataset", width = 375, height = 275)

if (interactive()) img
```
if (interactive() && capabilities("cairo")) {
  plotTag(
    plot(pressure), "A plot of the 'pressure' dataset",
    device = grDevices::svg, width = 375, height = 275, pixelratio = 1/72,
    mimeType = "image/svg+xml"
  )
}

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**

```r
## 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**

```r
renderDependencies(
  dependencies,
  srcType = c("href", "file"),
  encodeFunc = urlEncodePath,
  hrefFilter = identity
)
```
**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.

---

**renderDocument**

*Render an html_document object*

**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 shiny::createWebDependency() is used; it modifies the href and tells Shiny to serve a particular path on the filesystem.

**Value**

An HTML() string, with UTF-8 encoding.
renderTags

**Render tags into HTML**

**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()` for higher level rendering.)

**Usage**

```r
defoldTags(x, singletons = character(0), 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 render hooks, singletons, head, and dependency handling, and simply renders the given tag objects as HTML. Please use `renderTags()` if `x` has not already handled its dependencies and render hooks.

**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.
**resolveDependencies**  
*Resolve a list of dependencies*

**Description**

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.

---

**save_html**  
*Save an HTML object to a file*

**Description**

An S3 generic method for saving an HTML-like object to a file. The default method copies dependency files to the directory specified via `libdir`.

**Usage**

```r
save_html(html, file, ...)  
## Default S3 method:
save_html(html, file, background = "white", libdir = "lib", lang = "en", ...)
```

**Arguments**

- `html`: HTML content to print.
- `file`: File path or connection. If a file path containing a sub-directory, the sub-directory must already exist.
- `background`: Background color for web page.
- `...`: Further arguments passed to other methods.
singleton

libdir    Directory to copy dependencies to.
lang    Value of the <html> lang attribute.

singleton    Include content only once

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.
Value

surroundSingletons preprocesses 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(...)
tagAddRenderHook

Arguments

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

See Also

htmlTemplate() for more information about using HTML templates.
htmlDependency()

tagAddRenderHook tagAddRenderHook Modify a tag prior to rendering

Description

Adds a hook to call on a tag() object when it is rendered as HTML (with, for example, print(), renderTags(), as.tags(), etc).

Usage

tagAddRenderHook(tag, func, replace = FALSE)

Arguments

tag A tag() object.
func A function (hook) to call when the tag is rendered. This function should have at least one argument (the tag) and return anything that can be converted into tags via as.tags().
replace If TRUE, the previous hooks will be removed. If FALSE, func is appended to the previous hooks.

Details

The primary motivation for tagAddRenderHook() is to create tags that can change their attributes (e.g., change CSS classes) depending upon the context in which they’re rendered (e.g., use one set of CSS classes in one a page layout, but a different set in another page layout). In this situation, tagAddRenderHook() is preferable to tagFunction() since the latter is more a "black box" in the sense that you don’t know anything about the tag structure until it’s rendered.

Value

A tag() object with a .renderHooks field containing a list of functions (e.g. func). When the return value is rendered (such as with as.tags()), these functions will be called just prior to writing the HTML.

See Also

tagFunction()
Examples

# Have a place holder div and return a span instead
obj <- div("example", .renderHook = function(x) {
  x$name <- "span"
  x
})
obj$name # "div"
print(obj) # Prints as a `span`

# Add a class to the tag
# Should print a `span` with class "extra"
spanExtra <- tagAddRenderHook(obj, function(x) {
  tagAppendAttributes(x, class = "extra")
})
spanExtra

# Replace the previous render method
# Should print a `div` with class "extra"
divExtra <- tagAddRenderHook(obj, replace = TRUE, function(x) {
  tagAppendAttributes(x, class = "extra")
})
divExtra

# Add more child tags
spanExtended <- tagAddRenderHook(obj, function(x) {
  tagAppendChildren(x, " ", tags$strong("bold text"))
})
spanExtended

# Add a new html dependency
newDep <- tagAddRenderHook(obj, function(x) {
  fa <- htmlDependency("font-awesome", "4.5.0", c(href="shared/font-awesome"),
                       stylesheet = "css/font-awesome.min.css")
  attachDependencies(x, fa, append = TRUE)
})
# Also add a jqueryui html dependency
htmlDependencies(newDep) <- htmlDependency("jqueryui", "1.11.4", c(href="shared/jqueryui"),
                                         script = "jquery-ui.min.js")
# At render time, both dependencies will be found
renderTags(newDep)$dependencies

# Ignore the original tag and return something completely new.
newObj <- tagAddRenderHook(obj, function(x) {
  tags$p("Something else")
})
newObj

tagAppendAttributes  Append tag attributes
Description

Append (tagAppendAttributes()), check existence (tagHasAttribute()), and obtain the value (tagGet Attribute()) of HTML attribute(s).

Usage

tagAppendAttributes(tag, ..., .cssSelector = NULL)
tagHasAttribute(tag, attr)
tagGetAttribute(tag, attr)

Arguments

tag a tag object.
...
Attributes to append as named argument-value pairs. A named argument with an NA value is rendered as a boolean attribute (see example).
.cssSelector A character string containing a CSS selector for targeting particular (inner) tags of interest. At the moment, only a combination of type (e.g., div), class (e.g., .my-class), id (e.g., #myID), and universal (*) selectors within a given simple selector is supported. Note, if .cssSelector is used, the returned tags will have their $children fields flattened to a single list() via tagQuery().
attr The name of an attribute.

See Also

tagAppendChildren(), tagQuery()

Examples

html <- div(a())
tagAppendAttributes(html, class = "foo")
tagAppendAttributes(html, .cssSelector = "a", class = "bar")
tagAppendAttributes(html, contenteditable = NA)
tagHasAttribute(div(foo = "bar"), "foo")
tagGetAttribute(div(foo = "bar"), "foo")
Usage

tagAppendChild(tag, child, .cssSelector = NULL)
tagAppendChildren(tag, ..., .cssSelector = NULL, list = NULL)
tagSetChildren(tag, ..., .cssSelector = NULL, list = NULL)
tagInsertChildren(tag, after, ..., .cssSelector = NULL, list = NULL)

Arguments

tag a tag object.
child A child element to append to a parent tag.
.cssSelector A character string containing a CSS selector for targeting particular (inner) tags of interest. At the moment, only a combination of type (e.g., div), class (e.g., .my-class), id (e.g., #myID), and universal (*) selectors within a given simple selector is supported. Note, if .cssSelector is used, the returned tags will have their $children fields flattened to a single list() via tagQuery().
... a collection of child elements.
list Deprecated. Use !!! instead to splice into ....
after an integer value (i.e., subscript) referring to the child position to append after.

See Also
tagAppendAttributes(), tagQuery()

Examples

html <- div(a(), h1())
tagAppendChild(html, span())
tagAppendChildren(html, span(), p())
tagSetChildren(html, span(), p())
tagInsertChildren(html, after = 1, span(), p())
Usage

tagFunction(func)

Arguments

func a function with no arguments that returns HTML tags and/or dependencies.

Details

When possible, use `tagAddRenderHook()` to provide both a tag structure and utilize a render function.

See Also

`tagAddRenderHook()`

Examples

```r
myDivDep <- tagFunction(function() {
  if (isTRUE(getOption("useDep", TRUE))) {
    htmlDependency(
      name = "lazy-dependency",
      version = "1.0", src = ""
    )
  }
})
myDiv <- attachDependencies(div(), myDivDep)
renderTags(myDiv)
withr::with_options(list(useDep = FALSE), renderTags(myDiv))
```

Examples

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

tagQuery

Query and modify HTML tags

Description

[Experimental]

tagQuery() provides a jQuery inspired interface for querying and modifying tag() (and tagList()) objects.

Usage

tagQuery(tags)

Arguments

tags A tag(), tagList(), or list() of tags.

Value

A class with methods that are described below. This class can’t be used directly inside other tag() or a renderTags() context, but underlying HTML tags may be extracted via $allTags() or $selectedTags().

Altered Tag structure

For performance reasons, the input tag structure to tagQuery() will be altered into a consistently expected shape.

Some alterations include:

• tags flattening their $children fields into a single list()
• tags relocating any attribute html_dependency() to be located in $children’
• tagList()-like structures relocating any attribute html dependency to be a entry in its list structure.

While the resulting tag shape has possibly changed, tagQuery()'s' resulting tags will still render to the same HTML value (ex: renderTags()) and HTML dependencies (ex: findDependencies()).
tagQuery

Vignette

To get started with using tagQuery(), visit https://rstudio.github.io/htmltools/articles/tagQuery.html.

Methods

Unless otherwise stated, tagQuery() methods accept a character vector as input.

Query methods:
Query methods identify particular subsets of the root tag using CSS selectors (or R functions).

Children:
- $find(cssSelector): Get the descendants of each selected tag, filtered by a cssSelector.
- $children(cssSelector = NULL): Get the direct children of each selected tag, optionally filtered by a cssSelector.

siblings(cssSelector = NULL): Get the siblings of each selected tag, optionally filtered by a cssSelector.

Parents:
- $parent(cssSelector = NULL): Get the parent of each selected tag, optionally filtered by a cssSelector.
- $parents(cssSelector = NULL): Get the ancestors of each selected tag, optionally filtered by a cssSelector.
- $closest(cssSelector = NULL): For each selected tag, get the closest ancestor tag (including itself) satisfying a cssSelector. If cssSelector = NULL, it is equivalent to calling $selectedTags().

Custom filter:
- $filter(fn): Filter the selected tags to those for which fn(x, i) returns TRUE. In addition to an R function with two arguments (the selected tag x and the index i), fn may also be a valid CSS selector.

Length:
- $length(): Number of tags that have been selected.

Reset:
- $resetSelected(): Reset selected tags to the $allTags() tag. Useful in combination with $replaceWith() since it empties the selection.

Modify methods:
Unlike query methods, modify methods modify the tagQuery() object.

Attributes:
- $addClass(class): Adds class(es) to each selected tag.
- $removeClass(class): Removes class(es) to each selected tag.
- $toggleClass(class): Adds class(es) that don’t already exist and removes class(es) that do already exist (for each selected tag).
- $hasClass(class): Does each selected tag have all the provided class(es)?
- $addAttrs(...): Add a set of attributes to each selected tag.
$\text{removeAttrs}($\text{attrs}$)$: Remove a set of attributes from each selected tag.

$\text{hasAttrs}($\text{attr}$)$: Do each selected tags have all of the attributes?

**Children:**
- $\text{append}(\ldots)$: For each selected tag, insert \ldots after any existing children.
- $\text{prepend}(\ldots)$: For each selected tag, insert \ldots before any existing children.

**Siblings:**
- $\text{after}(\ldots)$: Add all \ldots objects as siblings after each of the selected tags.
- $\text{before}(\ldots)$: Add all \ldots objects as siblings before each of the selected tags.

**Custom:**
- $\text{each}(\text{fn})$: Modify each selected tag with a function \text{fn}. \text{fn} should accept two arguments: the first is the selected tag and second is the selected tags position index. Since the selected tag is a reference, any modifications to it will also modify the \text{tagQuery()} object.

**Replace methods:**
- $\text{replaceWith}(\ldots)$: Replace all selected tags with \ldots in the root tag and clear the selection.
- $\text{remove}(\ldots)$: Remove all selected tags from the root tag and clear the current selection.
- $\text{empty}()$: Remove any children of each selected tag. Use this method before calling $\text{append}(\ldots)$ to replace the children of each selected tag, with other content.

**Extract HTML tags:**
- $\text{allTags}()$: Return the (possibly modified) root tags.
- $\text{selectedTags}()$: Return a \text{tagList()} of the currently selected tags.

**Examples**

```r
tagQ <- tagQuery(div(a()))
tagQ$find("a")$addClass("foo")
tagQ
```

# To learn more, visit https://rstudio.github.io/htmltools/articles/tagQuery.html

---

**urlEncodePath**

**Encode a URL path**

**Description**

Encode characters in a URL path. This is the same as \text{utils::URLencode()} with \text{reserved} = \text{TRUE} except that / is preserved.

**Usage**

```r
urlEncodePath(x)
```

**Arguments**

- \text{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**

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", "fit-content" 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**

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

```r
withTags(code, .noWS = NULL)
```

Arguments

- **code**: A set of tags.
- **.noWS**: Default whitespace behavior for all tags within this call to `withTags()`. Setting `.noWS` on an individual tag function inside `withTags()` will override the default. See `tag()` for complete options.

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

```r
# 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")
    )
)

# Setting .noWS for all tags in withTags()
withTags(
    div(
        class = "myclass",
        h3("header"),
        p("One", strong(span("two")), "three")
    ),
```
withTags

  .noWS = c("outside", "inside")
}

Index

a (builder), 5
as.character(), 3, 22, 33
as.tags, 3
as.tags(), 6, 25, 29
attachDependencies (htmlDependencies), 12
attachDependencies(), 15
base::print(), 18
bindFillRole, 3
br (builder), 5
browsable, 4
browsable(), 22
builder, 5, 27

Cairo::CairoPNG(), 8, 11, 22
capturePlot, 7
capturePlot(), 22
code (builder), 5
col2rgb(), 21
copyDependencyToDir, 9
copyDependencyToDir(), 20
css, 10
defaultPngDevice, 11
defaultPngDevice(), 8, 22
div (builder), 5
doRenderTags (renderTags), 25

em (builder), 5
extractPreserveChunks (htmlPreserve), 16

findDependencies, 11
findDependencies(), 34

grDevices::png(), 8, 11, 22
grDevices::svg(), 22

h1 (builder), 5
h2 (builder), 5
h3 (builder), 5

h4 (builder), 5
h5 (builder), 5
h6 (builder), 5
hr (builder), 5
HTML, 12
HTML(), 6, 24, 25, 27
html_print, 18
htmlDependencies, 12
htmlDependencies(), 32
htmlDependencies<-(htmlDependencies), 12
htmlDependency, 13
htmlDependency(), 4, 6, 11, 24–26, 28, 29
htmlEscape, 15
htmlPreserve, 16
htmlTemplate, 17
htmlTemplate(), 24, 29

img (builder), 5
include, 18
includeCSS (include), 18
includeHTML (include), 18
includeMarkdown (include), 18
includeScript (include), 18
includeText (include), 18
is.browsable (browsable), 4
is.singleton (singleton), 27

knit_print.html (knitr_methods), 19
knit_print.html_dependency
(knitr_methods), 19
knit_print.shiny.tag (knitr_methods), 19
knitr_methods, 19

list(), 34

makeDependencyRelative, 20
makeDependencyRelative(), 9

p (builder), 5
parseCssColors, 20
INDEX

plotTag, 21
plotTag(), 8
pre (builder), 5
pre(), 19
print(), 29, 33
print.html (print.shiny.tag), 23
print.html(), 25
print.shiny.tag, 23

ragg::agg.png(), 8, 11, 22
renderDependencies(), 23
renderDocument(), 24
renderDocument(), 17
renderTags(), 24, 29, 34
resolved, 25
resolveDependencies(), 26
resolveDependencies(), 14
restorePreserveChunks (htmlPreserve), 16

save_html, 26
shiny::createWebDependency(), 24
singleton, 25, 27
singleton(), 27
singleton_tools, 27
span (builder), 5
strong (builder), 5
subtractDependencies, 28
suppressDependencies, 28
surroundSingletons (singleton_tools), 27
system.file(), 26

tag, 12, 31–33
tag (builder), 5
tag(), 3, 27, 29, 34, 38
tagAddRenderHook, 29
tagAddRenderHook(), 6, 29, 33
tagAppendAttributes, 30
tagAppendAttributes(), 4, 7, 32
tagAppendChild, 31
tagAppendChildren (tagAppendChild), 31
tagAppendChildren(), 31
tagFunction, 32
tagFunction(), 29
tagGetAttribute (tagAppendAttributes), 30
tagHasAttribute (tagAppendAttributes), 30
tagInsertChildren (tagAppendChild), 31
tagList(), 7, 13, 34, 36
tagQuery, 34
tagQuery(), 3, 7, 31, 32
tags, 32
tag (builder), 5
tagSetChildren (tagAppendChild), 31
takeSingletons (singleton_tools), 27
urlEncodePath, 36
utils::URLencode(), 36
validateCssUnit, 37
withTags, 38
withTags(), 7