Package 'htmltools'

September 8, 2014

Type Package

Title Tools for HTML

Version 0.2.6

Date 2014-07-30

Author RStudio, Inc.

Maintainer Joe Cheng <joe@rstudio.com>

Description Tools for HTML generation and output

Depends R (>= 2.14.1)

Imports utils, digest

Suggests markdown, testthat

Enhances knitr

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2014-09-08 19:23:28

R topics documented:

`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

```
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.

builder *HTML Builder Functions*

Description

Simple functions for constructing HTML documents.

Usage

```
tags

p(...)

h1(...)

h2(...)
```

```
h3(...)

h4(...)

h5(...)

h6(...)

a(...)

br(...)

div(...)

span(...)

pre(...)

code(...)

img(...)

strong(...)

em(...)

hr(...)
```

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), and raw HTML (see `HTML`). You can also pass lists that contain tags, text nodes, and HTML.

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()`.

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))
```

`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").

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.

If a subdirectory named *name-version* already exists in `outputDir`, then copying is not performed; the existing contents are assumed to be up-to-date.

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.

findDependencies *Collect attached dependencies from HTML tag object*

Description

Walks a hierarchy of tags looking for attached dependencies.

Usage

```
findDependencies(tags)
```

Arguments

tags	A tag-like object to search 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

```
HTML(text, ...)
```

Arguments

text	The text value to mark with HTML
...	Any additional values to be converted to character and concatenated together

Value

The same value, but marked as HTML.

Examples

```
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

```
htmlDependencies(x)

htmlDependencies(x) <- value

attachDependencies(x, value)
```

Arguments

x	An object which has (or should have) HTML dependencies.
value	An HTML dependency, or a list of HTML dependencies.

Details

`attachDependencies` provides an alternate syntax for setting dependencies and is essentially equivalent to `local({htmlDependencies(x) <- value; x})`.

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

```
htmlDependency(name, version, src, meta = NULL, script = NULL,
  stylesheet = NULL, head = NULL, attachment = NULL)
```

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 path parameter).
stylesheet	Stylesheet(s) to include within the document (should be specified relative to the path parameter).
head	Arbitrary lines of HTML to insert into the document head
attachment	Attachment(s) to include within the document head. See Details.

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 Note that depending on the rendering context, the runtime value of the href may be an absolute, relative, or data URI.

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

```
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
```

---

html_print *Implementation of the print method for HTML*

Description

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

Usage

```
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

```
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/rmarkdown 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

```
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 baspath 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

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)
```

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 stylsheet files. The default should generally be used.

Value

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

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(browse=TRUE) for higher level rendering.)

Usage

```
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.

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

```
resolveDependencies(dependencies)
```

Arguments

dependencies	A list of htmlDependency objects.

Value

dependencies A list of htmlDependency objects with redundancies removed.

save_html *Save an HTML object to a file*

Description

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

Usage

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

Arguments

html	HTML content to print
background	Background color for web page
file	File to write content to
libdir	Directory to copy dependenies to

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.

---

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. tagAppendChild() and tagList() 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, ...)

tagAppendChild(tag, child)

tagAppendChildren(tag, ..., list = NULL)

tagSetChildren(tag, ..., list = NULL)

tag(`_tag_name`, varArgs)
```

Arguments

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

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)
```

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

```
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"`, or a number. If the number has a suffix, it must be valid: `px`, `%`, `em`, `pt`, `in`, `cm`, `mm`, `ex`, or `pc`. 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

```
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