

# Package ‘htmltools’

April 28, 2017

**Type** Package

**Title** Tools for HTML

**Version** 0.3.6

**Date** 2017-04-26

**Author** RStudio, Inc.

**Maintainer** Joe Cheng <joe@rstudio.com>

**Description** Tools for HTML generation and output.

**Depends** R (>= 2.14.1)

**Imports** utils, digest, Rcpp

**Suggests** markdown, testthat

**Enhances** knitr

**License** GPL (>= 2)

**URL** <https://github.com/rstudio/htmltools>

**BugReports** <https://github.com/rstudio/htmltools/issues>

**RoxygenNote** 6.0.1

**LinkingTo** Rcpp

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2017-04-28 07:41:46 UTC

## R topics documented:

as.tags . . . . .	2
browsable . . . . .	3
builder . . . . .	3
copyDependencyToDir . . . . .	5
css . . . . .	6
findDependencies . . . . .	7
HTML . . . . .	7

htmlDependencies . . . . .	8
htmlDependency . . . . .	9
htmlEscape . . . . .	10
htmlPreserve . . . . .	11
htmlTemplate . . . . .	12
html_print . . . . .	13
include . . . . .	13
knitr_methods . . . . .	14
makeDependencyRelative . . . . .	15
print.shiny.tag . . . . .	15
renderDependencies . . . . .	16
renderDocument . . . . .	16
renderTags . . . . .	17
resolveDependencies . . . . .	18
save_html . . . . .	18
singleton . . . . .	19
singleton_tools . . . . .	19
subtractDependencies . . . . .	20
suppressDependencies . . . . .	20
tag . . . . .	21
urlEncodePath . . . . .	22
validateCssUnit . . . . .	23
withTags . . . . .	23

<b>Index</b>	<b>25</b>
--------------	-----------

---

as.tags	<i>Convert a value to tags</i>
---------	--------------------------------

---

## 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	<i>Make an HTML object browsable</i>
-----------	--------------------------------------

---

**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**

<code>x</code>	The object to make browsable or not.
<code>value</code>	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.

---

<code>builder</code>	<i>HTML Builder Functions</i>
----------------------	-------------------------------

---

**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), raw HTML (see [HTML](#)), and `html_dependency` objects. You can also pass lists that contain tags, text nodes, or 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(  
    <h1>Hello</h1>  
  )  
)
```

```

    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"). 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

<code>...</code>	Named style properties, where the name is the property name and the argument is the property value. See Details for conversion rules.
<code>collapse_</code>	(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
)
```

---

findDependencies	<i>Collect attached dependencies from HTML tag object</i>
------------------	---

---

**Description**

Walks a hierarchy of tags looking for attached dependencies.

**Usage**

```
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	<i>Mark Characters as HTML</i>
------	--------------------------------

---

**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**

```
e1 <- div(HTML("I like <u>turtles</u>"))  
cat(as.character(e1))
```

---

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, append = FALSE)
```

### 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) 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

```
# 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	<i>Define an HTML dependency</i>
----------------	----------------------------------

---

## 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, 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 <a href="#">resolveDependencies</a> ).
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)`. 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

<code>text</code>	Text to escape
<code>attribute</code>	Escape for use as an attribute value

## Value

Character vector with escaped text.

---

htmlPreserve	<i>Preserve HTML regions</i>
--------------	------------------------------

---

### 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

```

---

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

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

## Arguments

<code>filename</code>	Path to an HTML template file. Incompatible with <code>text_</code> .
<code>...</code>	Variable values to use when processing the template.
<code>text_</code>	A string to use as the template, instead of a file. Incompatible with <code>filename</code> .
<code>document_</code>	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	<i>Implementation of the print method for HTML</i>
------------	--

---

**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	<i>Include Content From a File</i>
---------	------------------------------------

---

**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 <code>knit_print</code>
...	Additional <code>knit_print</code> 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**

<code>dependency</code>	A single HTML dependency with an absolute path.
<code>basepath</code>	The path to the directory that dependency should be made relative to.
<code>mustWork</code>	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](#)

---

`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	<i>Create HTML for dependencies</i>
--------------------	-------------------------------------

---

**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 stylesheet files. The default should generally be used.

**Value**

An [HTML](#) object suitable for inclusion in the head of an HTML document.

---

renderDocument	<i>Render an html_document object</i>
----------------	---------------------------------------

---

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

---

renderTags	<i>Render tags into HTML</i>
------------	------------------------------

---

**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 <code>singleton</code> 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, 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**

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
file	File to write content to
background	Background color for web page
libdir	Directory to copy dependencies to

---

singleton	<i>Include content only once</i>
-----------	----------------------------------

---

**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 <a href="#">tag</a> , text, HTML, or list.
value	Whether the object should be a singleton.

---

singleton_tools	<i>Singleton manipulation functions</i>
-----------------	---

---

**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 <a href="#">builder</a> topic.
singletons	Character vector of singleton signatures that have already been encountered (i.e. returned from previous calls to <code>takeSingletons</code> ).
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(...)
```

**Arguments**

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

**See Also**

[htmlTemplate](#) for more information about using HTML templates.

[htmlDependency](#)

---

tag	<i>HTML Tag Object</i>
-----	------------------------

---

**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, ...)
```

```
tagHasAttribute(tag, attr)
```

```
tagGetAttribute(tag, attr)
```

```
tagAppendChild(tag, child)
```

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

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

```
tag(`_tag_name`, varArgs)
```

**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

`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.

### 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	<i>Validate proper CSS formatting of a unit</i>
-----------------	---

---

**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, 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	<i>Evaluate an expression using tags</i>
----------	--

---

**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), 3  
as.character, 2, 4, 22  
as.tags, 2, 17  
attachDependencies, 10  
attachDependencies (htmlDependencies), 8  
  
br (builder), 3  
browsable, 3  
builder, 3, 19  
  
code (builder), 3  
copyDependencyToDir, 5, 15  
createWebDependency, 17  
css, 6  
  
div (builder), 3  
doRenderTags (renderTags), 17  
  
em (builder), 3  
extractPreserveChunks (htmlPreserve), 11  
  
findDependencies, 7  
  
h1 (builder), 3  
h2 (builder), 3  
h3 (builder), 3  
h4 (builder), 3  
h5 (builder), 3  
h6 (builder), 3  
hr (builder), 3  
HTML, 4, 7, 16, 18, 19, 22  
html\_print, 13  
htmlDependencies, 8  
htmlDependencies<- (htmlDependencies), 8  
htmlDependency, 7, 9, 16–18, 20, 21  
htmlEscape, 10  
htmlPreserve, 11  
htmlTemplate, 12, 17, 21  
  
img (builder), 3  
include, 13  
  
includeCSS (include), 13  
includeHTML (include), 13  
includeMarkdown (include), 13  
includeScript (include), 13  
includeText (include), 13  
is.browsable (browsable), 3  
is.singleton (singleton), 19  
  
knit\_print.html (knitr\_methods), 14  
knit\_print.shiny.tag (knitr\_methods), 14  
knitr\_methods, 14  
  
makeDependencyRelative, 5, 15  
  
p (builder), 3  
pre, 14  
pre (builder), 3  
print, 13  
print.html, 17  
print.html (print.shiny.tag), 15  
print.shiny.tag, 15  
  
renderDependencies, 16  
renderDocument, 12, 16  
renderTags, 16, 17  
resolved, 18  
resolveDependencies, 9, 18  
restorePreserveChunks (htmlPreserve), 11  
  
save\_html, 18  
singleton, 17, 19, 19  
singleton\_tools, 19  
span (builder), 3  
strong (builder), 3  
subtractDependencies, 20  
suppressDependencies, 20  
surroundSingletons (singleton\_tools), 19  
system.file, 18  
  
tag, 4, 7, 19, 21  
tagAppendAttributes (tag), 21

tagAppendChild (tag), 21  
tagAppendChildren (tag), 21  
tagGetAttribute (tag), 21  
tagHasAttribute (tag), 21  
tagList, 8  
tagList (tag), 21  
tags, 21  
tags (builder), 3  
tagSetChildren (tag), 21  
takeSingletons (singleton\_tools), 19

URLencode, 22  
urlEncodePath, 22

validateCssUnit, 23

withTags, 23