Package 'markdown'

August 24, 2014

Type Package

Title Markdown rendering for R

Version 0.7.4

Date 2014-08-24

Author JJ Allaire, Jeffrey Horner, Vicent Marti, and Natacha Porte

Maintainer Yihui Xie <xie@yihui.name>

Description Markdown is a plain-text formatting syntax that can be converted to XHTML or other formats. This package provides R bindings to the Sundown markdown rendering library.

Depends R (>= 2.11.1)

Imports mime (>= 0.1.1)

Suggests knitr, RCurl

License GPL-2

URL https://github.com/rstudio/markdown

BugReports https://github.com/rstudio/markdown/issues

VignetteBuilder knitr

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-08-24 09:58:03

R topics documented:

`markdown` *Markdown rendering for R*

Description

Markdown is a plain-text formatting syntax that can be converted to XHTML or other formats. This package provides R bindings to the Sundown (`https://github.com/vmg/sundown`) markdown rendering library.

Details

The R function `markdownToHTML` renders a markdown file to HTML (respecting the specified `markdownExtensions` and `markdownHTMLOptions`).

The package also exports the underlying Sundown C extension API which enables creating and calling custom renderers using the `renderMarkdown` function.

To learn more about markdown syntax see:

`http://en.wikipedia.org/wiki/Markdown`

Author(s)

JJ Allaire, Jeffrey Horner, Vicent Marti, and Natacha Porte

Maintainer: Yihui Xie <xie@yihui.name>

See Also

`markdownToHTML` `renderMarkdown`

`markdownExtensions` *Markdown extensions*

Description

`markdownExtensions` returns a character vector listing all the extensions that are available in the **markdown** package.

Usage

```
markdownExtensions()
```

Details

They are all ON by default.

The **Sundown** library (upon which **markdown** is built) has optional support for several extensions described below. To turn these on globally in the **markdown** package, simply place some or all of them in a character vector and assign to the global option `markdown.extensions` like so:

```
options(markdown.extensions = markdownExtensions())
```

To override the global option, pass the `extensions` as an argument to one of the render functions, e.g.:

```
markdownToHTML(..., extensions = c('no_intra_emphasis'))
```

Description of all extensions:

`'no_intra_emphasis'` skip markdown embedded in words.

`'tables'` create HTML tables (see Examples).

`'fenced_code'` treat text as verbatim when surrounded with begin and ending lines with three ~ or ‘ characters.

`'autolink'` create HTML links from urls and email addresses.

`'strikethrough'` create strikethroughs by surrounding text with ~~.

`'lax_spacing'` allow HTML tags inside paragraphs without being surrounded by newlines.

`'space_headers'` add a space between header hashes and the header itself.

`'superscript'` translate ^ and subsequent text into HTML superscript.

`'latex_math'` transforms all math equations into syntactically correct MathJax equations.

See the EXAMPLES section to see the output of each extension turned on or off.

Value

A `character` vector listing all available extensions.

See Also

markdownHTMLOptions

Examples

```
# List all available extensions:
markdownExtensions()

# To turn on all markdown extensions globally:
options(markdown.extensions = markdownExtensions())

# To turn off all markdown extensions globally:
options(markdown.extensions = NULL)
# The following examples are short, so we set the HTML option 'fragment_only'

options(markdown.HTML.options = "fragment_only")

# no_intra_emphasis example
cat(markdownToHTML(text = "foo_bar_function", extensions = c()))
cat(markdownToHTML(text = "foo_bar_function", extensions = c("no_intra_emphasis")))

# tables example (need 4 spaces at beginning of line here)
cat(markdownToHTML(text = "
    First Header  | Second Header
    ------------- | -------------
    Content Cell  | Content Cell
    Content Cell  | Content Cell
", extensions = c()))

# but not here
cat(markdownToHTML(text = "
First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
", extensions = c("tables")))

# fenced_code example (need at least three leading ~ or `)
fenced_block <- function(text, x = "`", n = 3) {
  fence <- paste(rep(x, n), collapse = "")
  paste(fence, text, fence, sep = "")
}
cat(markdownToHTML(text = fenced_block("
preformatted text here without having to indent
first line.
"), extensions = c()))

cat(markdownToHTML(text = fenced_block("
preformatted text here without having to indent
first line.
"), extensions = c("fenced_code")))

# autolink example
cat(markdownToHTML(text = "http://www.r-project.org/", extensions = c()))
cat(markdownToHTML(text = "http://www.r-project.org/", extensions = c("autolink")))
```

```
# strikethrough example
cat(markdownToHTML(text = "~~awesome~~", extensions = c()))
cat(markdownToHTML(text = "~~awesome~~", extensions = c("strikethrough")))

# lax_spacing
cat(markdownToHTML(text = "
Embedding html without surrounding with empty newline.
<div>_markdown_</div>
extra text.
", extensions = c("")))
cat(markdownToHTML(text = "
Embedding html without surrounding with empty newline.
<div>_markdown_</div>
extra text.
", extensions = c("lax_spacing")))

# space_headers example
cat(markdownToHTML(text = "#A Header\neven though there is no space between # and A",
                   extensions = c("")))
cat(markdownToHTML(text = "#Not A Header\nbecause there is no space between # and N",
                   extensions = c("space_headers")))

# superscript example
cat(markdownToHTML(text = "2^10", extensions = c()))
cat(markdownToHTML(text = "2^10", extensions = c("superscript")))
```

`markdownHTMLOptions` *Markdown HTML rendering options*

Description

`markdownHTMLOptions` returns a character vector listing all the options that are available for the HTML renderer in the **markdown** package. As a convenience, the package default options were chosen to render well-formed stand-alone HTML pages when using `markdownToHTML()`. The default options are `'use_xhtml'`, `'smartypants'`, `'base64_images'`, `'mathjax'`, and `'highlight_code'`.

Usage

```
markdownHTMLOptions(defaults = FALSE)
```

Arguments

`defaults`	If `TRUE`, then only the default options are returned. Otherwise all options are returned.

Details

The HTML renderer provides several options described below. To turn these on globally in the **markdown** package, simply place some or all of them in a character vector and assign to the global option `markdown.HTML.options` like so:

```
options(markdown.HTML.options = markdownHTMLOptions())
```

To reset the options to package default, use:

```
options(markdown.HTML.options = markdownHTMLOptions(default = TRUE))
```

To override the global option, pass the `options` as an argument:

```
markdownToHTML(..., options = c('skip_images'))
```

Description of all options:

`'skip_html'` suppress output of all HTML tags in the document.

`'skip_style'` suppress output of HTML style tags.

`'skip_images'` suppress output of HTML image tags.

`'skip_links'` suppress output of HTML anchor tags.

`'safelink'` only create links for known url types, e.g. http, ftp, http, etc.

`'toc'` assigns an HTML id to each header of the form 'toc_ where ' (starting at 0), and creates the table of contents.

`'hard_wrap'` adds an HTML br tag for every newline (excluding trailing) found within a paragraph.

`'use_xhtml'` create XHMTL 1.0 compliant HTML tags.

`'escape'` escape all HTML found within the *markdown*. Overrides all of the `'skip_*'` options mentioned above.

`'smartypants'` translates plain ASCII punctuation characters into *smart* typographic punctuation HTML entities.

`'fragment_only'` eliminates the inclusion of any HTML header or body tags, CSS, or Javascript components.

`'base64_images'` Any local images linked with the `'<img>'` tag to the output HTML will automatically be converted to base64 and included along with output.

`'mathjax'` includes appropriate Javascript libraries to render math markup.

`'highlight_code'` includes appropriate Javascript libraries to highlight code chunks.

See the EXAMPLES section to see the output of each option turned on or off.

Value

A `character` vector listing either all available options or just the default options.

See Also

markdownToHTML

Examples

```
# List all available extensions:
markdownHTMLOptions()

# To turn on all HTML options globally:
options(markdown.HTML.options = markdownHTMLOptions())

# To turn off all HTML options globally:
options(markdown.HTML.options = NULL)

# To turn on package default HTML options globally:
options(markdown.HTML.options = markdownHTMLOptions(default = TRUE))
# HTML OPTIONS

# The following examples are short, so we allways add the HTML option 'fragment_only'
tOpt <- "fragment_only"

# skip_html example
mkd = '<style></style><img src="http://cran.rstudio.com/Rlogo.jpg"><a href="#">Hello</a>'
cat(markdownToHTML(text = mkd, options = c(tOpt)))
cat(markdownToHTML(text = mkd, options = c(tOpt, "skip_html")))

# skip_style example
cat(markdownToHTML(text = mkd, options = c(tOpt)))
cat(markdownToHTML(text = mkd, options = c(tOpt, "skip_style")))

# skip_images example
cat(markdownToHTML(text = mkd, options = c(tOpt)))
cat(markdownToHTML(text = mkd, options = c(tOpt, "skip_images")))

# skip_links example
cat(markdownToHTML(text = mkd, options = c(tOpt)))
cat(markdownToHTML(text = mkd, options = c(tOpt, "skip_links")))

# safelink example
cat(markdownToHTML(text = '[foo](foo://bar "baz")', options = c(tOpt)))
cat(markdownToHTML(text = '[foo](foo://bar "baz")', options = c(tOpt, "safelink")))

# toc example
mkd <- paste(c("# Header 1", "p1", "## Header 2", "p2"), collapse = "\n")

cat(markdownToHTML(text = mkd, options = c(tOpt)))
cat(markdownToHTML(text = mkd, options = c(tOpt, "toc")))

# hard_wrap example
cat(markdownToHTML(text = "foo\nbar\n", options = c(tOpt)))
cat(markdownToHTML(text = "foo\nbar\n", options = c(tOpt, "hard_wrap")))

# use_xhtml example
cat(markdownToHTML(text = "foo\nbar\n", options = c(tOpt, "hard_wrap")))
cat(markdownToHTML(text = "foo\nbar\n", options = c(tOpt, "hard_wrap", "use_xhtml")))
```

```
# escape example
mkd = '<style></style><img src="http://cran.rstudio.com/Rlogo.jpg"><a href="#">Hello</a>'
cat(markdownToHTML(text = mkd, options = c(tOpt, "skip_html")))
# overrides all 'skip_*' options
cat(markdownToHTML(text = mkd, options = c(tOpt, "skip_html", "escape")))

# smartypants example
cat(markdownToHTML(text = "1/2 (c)", options = c(tOpt)))
cat(markdownToHTML(text = "1/2 (c)", options = c(tOpt, "smartypants")))

cat(smartypants(text = "1/2 (c)\n"))
```

`markdownToHTML` *Render markdown to HTML*

Description

`markdownToHTML` transforms the *markdown* text provided by the user in either the `file` or `text` variable. The HTML transformation is either written to the `output` file or returned to the user as a `character` vector.

Usage

```
markdownToHTML(file, output = NULL, text = NULL,
  options = getOption("markdown.HTML.options"),
  extensions = getOption("markdown.extensions"), title = "",
  stylesheet = getOption("markdown.HTML.stylesheet"),
  header = getOption("markdown.HTML.header"),
  template = getOption("markdown.HTML.template"), fragment.only = FALSE,
  encoding = getOption("encoding"))
```

Arguments

`file`	a character string giving the pathname of the file to read from. If it is omitted from the argument list, then it is presumed that the `text` argument will be used instead.
`output`	a character string giving the pathname of the file to write to. If it is omitted (`NULL`), then it is presumed that the user expects the results returned as a `character` vector.
`text`	a character vector containing the *markdown* text to transform (each element of this vector is treated as a line in a file).
`options`	options that are passed to the renderer. see `markdownHTMLOptions`.
`extensions`	options that are passed to the *markdown* engine. See `markdownExtensions`.
`title`	The HTML title.
`stylesheet`	either valid CSS or a file containing CSS. will be included in the output.

header	either valid HTML or a file containing HTML will be included in the header of the output.
template	an HTML file used as template.
fragment.only	Whether or not to produce an HTML fragment without the HTML header and body tags, CSS, and Javascript components.
encoding	the encoding of the input file; see file

Details

Three notable HTML options have been added to support collaborative reproducible research. They are as follows:

- Latex math expressions enclosed by one of the block level syntaxes, $latex ... $, $$ ... $$, or \[... \], or one of the inline syntaxes, $...$, or \(... \), will be rendered in real-time by the MathJax Javascript library.
- *R* code blocks enclosed between ```r and ``` will automatically be syntax highlighted.
- Any local images linked using the <img> tag will be base64 encoded and included in the output HTML.

See the DETAILS section below and markdownHTMLOptions for more information.

There are two basic modes to markdownToHTML determined by the value of the fragment.only argument:

When FALSE, markdownToHTML creates well-formed stand-alone HTML pages complete with HTML header, title, and body tags. The default template used for this mode may be found here:

```
system.file('resources', 'markdown.html', package = 'markdown')
```

Also, markdownToHTML will automatically determine whether or not mathjax and R code highlighting are needed and will include the appropriate Javascript libraries in the output. Thus, there's no need to explicitly set the 'mathjax' or 'highlight_code' options (see markdownHTMLOptions for more details).

When fragment.only is TRUE, nothing extra is added.

Value

Invisible NULL when output is to a file, and a character vector otherwise.

See Also

markdownExtensions, markdownHTMLOptions, renderMarkdown.

Examples

```
(markdownToHTML(text = "Hello World!", fragment.only = TRUE))
(markdownToHTML(file = NULL, text = "_text_ will override _file_", fragment.only = TRUE))
# write HTML to an output file
markdownToHTML(text = "_Hello_, **World**!", output = "test.html")
```

registeredRenderers *List of Registered Markdown Renderers*

Description

registeredRenderers returns a named character vector listing all the registered renderers known to the **markdown** package. **markdown** allows up to seven renderers to be registered by users; HTML is provided by the package.

Usage

```
registeredRenderers()
```

Value

A named character vector listing all available renderers. Vector value contain renderer names, and named values contain the renderer output type, either character or raw.

See Also

markdownToHTML, rendererOutputType

Examples

```
# List all available renderers
registeredRenderers()
```

rendererExists *Testing for existence of a markdown renderer*

Description

rendererExists determines whether or not a certain renderer exists in the markdown library.

Usage

```
rendererExists(name)
```

Arguments

name	name of renderer.

Value

TRUE or FALSE for whether or not the renderer exists.

Examples

```
rendererExists("HTML")
```

rendererOutputType *Fetch the Renderer Output Type*

Description

markdown allows up to seven renderers to be registered by users, and each must provide the type of output returned, either `character` or `raw` for binary output. HTML is provided by the package and outputs `character`.

Usage

```
rendererOutputType(name)
```

Arguments

`name`	a character string naming the renderer.

Value

The character string with a value of either `character` or `raw`.

See Also

markdownToHTML, registeredRenderers

Examples

```
# List all available renderers
rendererOutputType("HTML")
```

renderMarkdown *Render markdown to an HTML fragment*

Description

`renderMarkdown` transforms the *markdown* text provided by the user in either the `file` or `text` variable. The transformation is either written to the `output` file or returned to the user. The default rendering target is "HTML".

Usage

```
renderMarkdown(file, output = NULL, text = NULL, renderer = "HTML",
    renderer.options = NULL, extensions = getOption("markdown.extensions"))
```

Arguments

renderer	the name of the renderer that will be used to transform the file or text.
renderer.options	options that are passed to the renderer. For HTML renderer options see markdownHTMLOptions.
file	a character string giving the pathname of the file to read from. If it is omitted from the argument list, then it is presumed that the text argument will be used instead.
output	a character string giving the pathname of the file to write to. If it is omitted (NULL), then it is presumed that the user expects the results returned as a character vector.
text	a character vector containing the *markdown* text to transform (each element of this vector is treated as a line in a file).
extensions	options that are passed to the *markdown* engine. See markdownExtensions.

Details

markdown uses (and ships with) the popular Sundown library provided by GitHub. C stubs are available to implement new renderers.

Value

renderMarkdown returns NULL invisibly when output is to a file, and either character or raw vector depending on the renderer output type.

See Also

markdownExtensions, markdownHTMLOptions, markdownToHTML.

For a description of the original *markdown* version:

http://daringfireball.net/projects/markdown/

The original Sundown library on github:

https://github.com/vmg/sundown

C stubs for writing new renders are in inst/include/markdown_rstubs.[ch].

Examples

```
renderMarkdown(text = "Hello World!")
```

rpubsUpload *Upload an HTML file to RPubs*

Description

This function uploads an HTML file to rpubs.com. If the upload succeeds a list that includes an `id` and `continueUrl` is returned. A browser should be opened to the `continueUrl` to complete publishing of the document. If an error occurs then a diagnostic message is returned in the `error` element of the list.

Usage

```
rpubsUpload(title, htmlFile, id = NULL, properties = list(),
    method = getOption("rpubs.upload.method", "auto"))
```

Arguments

`title`	The title of the document.
`htmlFile`	The path to the HTML file to upload.
`id`	If this upload is an update of an existing document then the id parameter should specify the document id to update. Note that the id is provided as an element of the list returned by successful calls to `rpubsUpload`.
`properties`	A named list containing additional document properties (RPubs doesn't currently expect any additional properties, this parameter is reserved for future use).
`method`	Method to be used for uploading. "internal" uses a plain http socket connection; "curl" uses the curl binary to do an https upload; "rcurl" uses the RCurl package to do an https upload; and "auto" uses the best available method searched for in the following order: "curl", "rcurl", and then "internal". The global default behavior can be configured by setting the `rpubs.upload.method` option (the default is "auto").

Value

A named list. If the upload was successful then the list contains a `id` element that can be used to subsequently update the document as well as a `continueUrl` element that provides a URL that a browser should be opened to in order to complete publishing of the document. If the upload fails then the list contains an `error` element which contains an explanation of the error that occurred.

Examples

```
## Not run:
# upload a document
result <- rpubsUpload("My document title", "Document.html")
if (!is.null(result$continueUrl))
    browseURL(result$continueUrl) else stop(result$error)

# update the same document with a new title
```

```
updateResult <- rpubsUpload("My updated title", "Document.html", result$id)

## End(Not run)
```

`smartypants` *smartypants: ASCII punctuation to HTML entities*

Description

`smartypants` transforms plain ASCII punctuation characters into *smart* typographic punctuation HTML entities.

Usage

```
smartypants(file, output, text)
```

Arguments

`file`	a character string giving the pathname of the file to read from. If it is omitted from the argument list, then it is presumed that the `text` argument will be used instead.
`output`	a character string giving the pathname of the file to write to. If it is omitted, then it is presumed that the user expects the results returned as a character string.
`text`	a character vector containing the *markdown* text to transform.

Value

`smartypants` returns NULL invisibly when output is to a file, and a character string otherwise.

See Also

`markdownExtensions`, `markdownHTMLOptions`, `markdownToHTML`.

For a description of the original *markdown* version:

http://daringfireball.net/projects/markdown/

The original Sundown library on github:

https://github.com/vmg/sundown

C stubs for writing new renders are in inst/include/markdown_rstubs.[ch].

Examples

```
cat(smartypants(text = "1/2 (c)\n"))
```

Index