Package ‘markdown’

November 16, 2022

Type Package

Title Render Markdown with 'commonmark'

Version 1.4

Description Render Markdown to full HTML documents with the 'commonmark'
package. Markdown is a plain-text formatting
syntax that can be converted to 'XHTML' or other formats. See
<https://en.wikipedia.org/wiki/Markdown> for more information about Markdown.

Depends R (>= 2.11.1)

Imports utils, commonmark, xfun, mime (>= 0.3)

Suggests knitr, rmarkdown (>= 2.18), yaml, RCurl

License GPL-2

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

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

VignetteBuilder knitr

RoxygenNote 7.2.1

Encoding UTF-8

NeedsCompilation no

Author JJ Allaire [aut],
Jeffrey Horner [aut],
Yihui Xie [aut, cre] (<https://orcid.org/0000-0003-0645-5666>),
Henrik Bengtsson [ctb],
Jim Hester [ctb],
Yixuan Qiu [ctb],
Kohske Takahashi [ctb],
Adam November [ctb],
Nacho Caballero [ctb],
Jeroen Ooms [ctb],
Thomas Leeper [ctb],
Joe Cheng [ctb],
Andrzej Oles [ctb],
RStudio [cph]

Maintainer Yihui Xie <xie@yihui.name>

Repository CRAN

Date/Publication 2022-11-16 12:10:42 UTC

R topics documented:

`html_format` *R Markdown output formats*

Description

Convenience functions for R Markdown v2 users.

Usage

```
html_format(
  meta = NULL,
  template = NULL,
  options = NULL,
  keep_md = FALSE,
  keep_tex = FALSE,
  latex_engine = "xelatex"
)

latex_format(
  meta = NULL,
  template = NULL,
  options = NULL,
  keep_md = FALSE,
  keep_tex = FALSE,
  latex_engine = "xelatex"
)
```

Arguments

`meta, template, options`	Arguments to be passed to `mark()`.
`keep_md, keep_tex`	Whether to keep the intermediate '`.md`' and '`.tex`' files generated from '`.Rmd`'.
`latex_engine`	The LaTeX engine to compile '`.tex`' to '`.pdf`'. This argument and `keep_tex` are for `latex_format()` only, and ignored in `html_format()`.

Details

We refer to this **markdown** package plus **knitr** as "R Markdown v1", and the **rmarkdown** package as "R Markdown v2". The former uses **commonmark** to convert Markdown, and the latter uses Pandoc. However, the latter also accept custom converting tools in addition to Pandoc. The output formats here provide the custom converting function `mark()` to **rmarkdown**, so that users can take advantage of `rmarkdown::render()` and the Knit button in RStudio. It is absolutely not necessary to rely on **rmarkdown**. The only point is convenience. If you do not use `rmarkdown::render()` or the Knit button, you can definitely just call `markdown::mark()` directly.

`mark` *Render Markdown to an output format*

Description

Render Markdown to an output format via the **commonmark** package. The function `mark_html()` is a shorthand of `mark(format = 'html')`, and `mark_latex()` is a shorthand of `mark(format = 'latex')`.

Usage

```
mark(
  file = NULL,
  output = NULL,
  text = NULL,
  format = c("html", "latex"),
  options = NULL,
  template = FALSE,
  meta = list(),
  ...
)

mark_html(..., options = NULL, template = TRUE, meta = list())

mark_latex(..., template = TRUE)
```

Arguments

`file`	Path to an input file. If not provided, it is presumed that the `text` argument will be used instead. This argument can also take a character vector of Markdown text directly. To avoid ambiguity in the latter case, a single character string input will be treated as a file only when the file exists or it has a file extension. If a string happens to have a “file extension” and should be treated as Markdown text instead, wrap it in `I()`.
`output`	Output file path. If not character, the results will be returned as a character vector.
`text`	A character vector of the Markdown text. By default, it is read from `file`.
`format`	An output format supported by **commonmark**, e.g., `'html'`, `'man'`, and `'text'`, etc. See the `markdown_*` renderers in **commonmark**.
`options`	Options to be passed to the renderer. See `markdown_options()` for all possible options. This argument can take either a character vector of the form `"+option1 option2-option3"` (use `+` or a space to enable an option, and `-` to disable an option), or a list of the form `list(option1 = value1, option2 = value2, ...)`. A string `"+option1"` is equivalent to `list(option1 = TRUE)`, and `"-option2"` means `list(option2 = FALSE)`. Options that do not take logical values must be specified via a list, e.g., `list(width = 30)`.
`template`	Path to a template file. The default value is `getOption('markdown.FORMAT.template',markdown:::pk` `'markdown.FORMAT'))` where `FORMAT` is the output format name (`html` or `latex`). It can also take a logical value: `TRUE` means to use the default template, and `FALSE` means to generate only a fragment without using any template.
`meta`	A named list of metadata. Elements in the metadata will be used to fill out the template by their names and values, e.g., `list(title = ...)` will replace the `$title$` variable in the template. See the ‘Details’ section for supported variables.
`...`	Arguments to be passed to `mark()`. For `mark_html()`, also additional arguments for backward-compatibility with previous versions of **markdown**. These are no longer recommended. For example, the `stylesheet` argument should be replaced by the `css` variable in `meta`, and the `fragment.only = TRUE` argument should be specified via `options = '-standalone'` instead.

Details

Supported variables in metadata for both HTML and HTML templates (the string `FORMAT` below is the output format name, i.e., `html` or `latex`):

`header-includes`, `include-before`, `include-after` Either a vector of code (HTML/LaTeX) or a code file to be included in the header, before the body, or after the body of the output. For `header-include`, the default value is taken from `getOption('markdown.FORMAT.header')` if not provided in `meta`.

`title` The document title.

Variables for the HTML template:

`css` Either a vector of CSS code or a file containing CSS to be included in the output. The default value is `getOption('markdown.html.css', markdown:::pkg_file('resources','markdown.css'))`, i.e., it can be set via the global option `markdown.html.css`.

`highlight` JavaScript code for syntax-highlighting code blocks. By default, the highlight.js library is used.

`math` JavaScript code for rendering LaTeX math. By default, MathJax is used.

Variables for the LaTeX template:

`classoption` A string containing options for the document class.

`documentclass` The document class (by default, `'article'`).

Note that you can use either underscores or hyphens in the variable names. Underscores will be normalized to hyphens internally, e.g., `header_includes` will be converted to `header-includes`. This means if you use a custom template, you must use hyphens instead of underscores as separators in variable names in the template.

Value

Invisible `NULL` when output is to a file, otherwise a character vector of the rendered output.

See Also

The spec of GitHub Flavored Markdown: `https://github.github.com/gfm/`

Examples

```
library(markdown)
mark(c("Hello _World_!", "", "Welcome to **markdown**."))
# a few corner cases
mark(character(0))
mark("")
# if input looks like file but should be treated as text, use I()
mark(I("This is *not* a file.md"))
# that's equivalent to
mark(text = "This is *not* a file.md")

mark_html("Hello _World_!", options = "-standalone")
# write HTML to an output file
mark_html("_Hello_, **World**!", output = tempfile())

mark_latex("Hello _World_!", template = FALSE)
```

`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 wrapper functions (mainly `mark()`) based on the **commonmark** package.

markdown_options *Markdown rendering options*

Description

A list of all available options to control Markdown rendering. Options that are enabled by default are marked by a `+` prefix, and those disabled by default are marked by `-`.

Usage

```
markdown_options()
```

Details

Description of all options:

- `base64_images` Embed local images in the HTML output with base64 encoding.
- `highlight_code` Includes JavaScript libraries to syntax highlight code blocks.
- `latex_math` Identify LaTeX math expressions in pairs of single or double dollar signs, and transform them so that they could be correctly rendered by MathJax (HTML output) or LaTeX.
- `mathjax` Include MathJax library in HTML output.
- `mathjax_embed` Whether to download and embed the MathJax library in HTML output.
- `smartypants` Translate certain ASCII strings into smart typographic characters (see `smartypants()`.
- `standalone` Generate a full (HTML/LaTeX) document or only a fragment of the body.
- `superscript` Translate strings between two carets into superscripts, e.g., `text^foo^` to `text<sup>foo</sup>`.
- `subscript` Translate strings between two tildes into subscripts, e.g., `text~foo~` to `text<sub>foo</sub>`.
- `toc` Generate a table of contents from section headers.
- `toc_depth` The number of section levels to include in the table of contents (`3` by default).
- `top_level` The desired type of the top-level headings in LaTeX output. Possible values are `'chapter'` and `'part'`. For example, if `top_level = 'chapter'`, `# header` will be rendered to `\chapter{header}` instead of the default `\section{header}`.

Options not described above can be found on the help pages of **commonmark**, e.g., the `hardbreaks` option is for the `hardbreaks` argument of `markdown_*()` functions, and the `table` option is for the `table` extension in **commonmark**'s extensions (`commonmark::list_extensions()`).

Value

A character vector of all available options.

Examples

```
# List all available options
markdown::markdown_options()

# Turn on/off some options globally for HTML output
options(markdown.html.options = '+toc-smartypants-standalone')

library(markdown)

# toc example
mkd <- c("# Header 1", "p1", "## Header 2", "p2")

cat(mark(mkd))
cat(mark(mkd, options = "toc"))

# hard_wrap example
cat(mark("foo\nbar\n"))
cat(mark("foo\nbar\n", options = "hard_wrap"))

# latex math example
mkd <- c(
  "`$x$` is inline math $x$!", "", "Display style:", "", "$$x + y$$", "",
  "\\begin{eqnarray}
a^{2}+b^{2} & = & c^{2}\\\\
\\sin^{2}(x)+\\cos^{2}(x) & = & 1
\\end{eqnarray}"
)

cat(mark(mkd))
cat(mark(mkd, options = "-latex_math"))

# tables example (need 4 spaces at beginning of line here)
cat(mark("
First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
"))

# but not here
cat(mark("
First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
", options = '-table'))

# autolink example
cat(mark("https://www.r-project.org/"))
cat(mark("https://www.r-project.org/", options = "-autolink"))

# strikethrough example
```

```
cat(mark("~~awesome~~"))
cat(mark("~~awesome~~", options = "-strikethrough"))

# superscript and subscript examples
cat(mark("2^10^"))
cat(mark("2^10^", options = "-superscript"))
cat(mark("H~2~O"))
cat(mark("H~2~O", options = "-subscript"))

# skip_html tags
mkd = '<style>a {}</style><script type="text/javascript">console.log("No!");</script>\n[Hello](#)'
cat(mark(mkd))
# TODO: wait for https://github.com/r-lib/commonmark/issues/15 to be fixed
# cat(mark(mkd, options = "tagfilter"))
```

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` *Convert some ASCII strings to HTML entities*

Description

Transform ASCII strings `(c)` (copyright), `(r)` (registered trademark), `(tm)` (trademark), and fractions `n/m` into *smart* typographic HTML entities.

Usage

```
smartypants(text)
```

Arguments

`text`	A character vector of the Markdown text.

Value

A character vector of the transformed text.

Examples

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

Index