Package 'litedown'

February 27, 2025

Type Package

Title A Lightweight Version of R Markdown

Version 0.6

Description Render R Markdown to Markdown (without using 'knitr'), and Markdown to lightweight HTML or 'LaTeX' documents with the 'commonmark' package (instead of 'Pandoc'). Some missing Markdown features in 'commonmark' are also supported, such as raw HTML or 'LaTeX' blocks, 'LaTeX' math, superscripts, subscripts, footnotes, element attributes, and appendices, but not all 'Pandoc' Markdown features are (or will be) supported. With additional JavaScript and CSS, you can also create HTML slides and articles. This package can be viewed as a trimmed-down version of R Markdown and 'knitr'. It does not aim at rich Markdown features or a large variety of output formats (the primary formats are HTML and 'LaTeX'). Book and website projects of multiple input documents are also supported.

Depends R (>= 3.2.0)

Imports utils, commonmark (>= 1.9.1), xfun (>= 0.51)

Suggests rbibutils, rstudioapi, tinytex

License MIT + file LICENSE

URL https://github.com/yihui/litedown

BugReports https://github.com/yihui/litedown/issues

VignetteBuilder litedown

RoxygenNote 7.3.2

Encoding UTF-8

NeedsCompilation no

Author Yihui Xie [aut, cre] (<https://orcid.org/0000-0003-0645-5666>, https://yihui.org), Tim Taylor [ctb] (<https://orcid.org/0000-0002-8587-7113>)

Maintainer Yihui Xie <xie@yihui.name>

Repository CRAN

Date/Publication 2025-02-27 09:40:02 UTC

Contents

`litedown-package` *A lightweight version of R Markdown*

Description

Markdown is a plain-text format that can be converted to HTML and other formats. This package can render R Markdown to Markdown, and then to an output document format. The main differences between this package and **rmarkdown** are that it does not use Pandoc or **knitr** (i.e., fewer dependencies), and it also has fewer Markdown features.

Author(s)

Maintainer: Yihui Xie <xie@yihui.name> (ORCID) (https://yihui.org)

Other contributors:

- Tim Taylor (ORCID) [contributor]

See Also

Useful links:

- https://github.com/yihui/litedown
- Report bugs at https://github.com/yihui/litedown/issues

crack *Parse R Markdown or R scripts*

Description

Parse input into code chunks, inline code expressions, and text fragments: `crack()` is for parsing R Markdown, and `sieve()` is for R scripts.

Usage

```
crack(input, text = NULL)

sieve(input, text = NULL)
```

Arguments

`input`	A character vector to provide the input file path or text. If not provided, the `text` argument must be provided instead. The `input` vector will be treated as a file path if it is a single string, and points to an existing file or has a filename extension. In other cases, the vector will be treated as the `text` argument input. To avoid ambiguity, if a string should be treated as `text` input when it happens to be an existing file path or has an extension, wrap it in `I()`, or simply use the `text` argument instead.
`text`	A character vector as the text input. By default, it is read from the `input` file if provided.

Details

For R Markdown, a code chunk must start with a fence of the form ```` ```{lang} ````, where `lang` is the language name, e.g., `r` or `python`. The body of a code chunk can start with chunk options written in "pipe comments", e.g., `#| eval = TRUE, echo = FALSE` (the CSV syntax) or `#| eval: true` (the YAML syntax). An inline code fragment is of the form `` `{lang} source` `` embedded in Markdown text.

For R scripts, text blocks are extracted by removing the leading `#'` tokens. All other lines are treated as R code, which can optionally be separated into chunks by consecutive lines of `#|` comments (chunk options are written in these comments). If no `#'` or `#|` tokens are found in the script, the script will be divided into chunks that contain smallest possible complete R expressions.

Value

A list of code chunks and text blocks:

- Code chunks are of the form `list(source, type = "code_chunk", options, comments, ...)`: `source` is a character vector of the source code of a code chunk, `options` is a list of chunk options, and `comments` is a vector of pipe comments.

- Text blocks are of the form `list(source, type = "text_block", ...)`. If the text block does not contain any inline code, `source` will be a character string (lines of text concatenated by line breaks), otherwise it will be a list with members that are either character strings (normal text fragments) or lists of the form `list(source, options, ...)` (`source` is the inline code, and `options` contains its options specified inside `` `{lang, ...}` ``).

Both code chunks and text blocks have a list member named `lines` that stores their starting and ending line numbers in the input.

Note

For simplicity, `sieve()` does not support inline code expressions. Text after `#'` is treated as pure Markdown.

It is a pure coincidence that the function names `crack()` and `sieve()` weakly resemble Carson Sievert's name, but I will consider adding a class name `sievert` to the returned value of `sieve()` if Carson becomes the president of the United States someday, which may make the value radioactive and introduce a new programming paradigm named *Radioactive Programming* (in case *Reactive Programming* is no longer fun or cool).

Examples

```
library(litedown)
# parse R Markdown
res = crack(c("```{r}\n1+1\n```", "Hello, `pi` = `{r} pi` and `e` = `{r} exp(1)`!"))
str(res)
# evaluate inline code and combine results with text fragments
txt = lapply(res[[2]]$source, function(x) {
    if (is.character(x))
        x else eval(parse(text = x$source))
})
paste(unlist(txt), collapse = "")

# parse R code
res = sieve(c("#' This is _doc_.", "", "#| eval=TRUE", "# this is code", "1 + 1"))
str(res)
```

`engines` *Language engines*

Description

Get or set language engines for evaluating code chunks and inline code.

Usage

```
engines(...)
```

Arguments

`...`	Named values (for setting) or unnamed values (for getting).

Details

An engine function should have three arguments:

- `x`: An element in the `crack()` list (a code chunk or a text block).
- `inline`: It indicates if `x` is from a code chunk or inline code.
- `...`: Currently unused but recommended for future compatibility (more arguments might be passed to the function).

The function should return a character value.

Value

The usage is similar to `reactor()`: `engines('LANG')` returns an engine function for the language `LANG`, and `engines(LANG = function(x, inline = FALSE, ...) {})` sets the engine for a language.

Examples

```
litedown::engines()  # built-in engines
```

`fuse` *Render Markdown, R Markdown, and R scripts*

Description

The function `fuse()` extracts and runs code from code chunks and inline code expressions in R Markdown, and interweaves the results with the rest of text in the input, which is similar to what `knitr::knit()` and `rmarkdown::render()` do. It also works on R scripts in a way similar to `knitr::spin()`. The function `fiss()` extracts code from the input, and is similar to `knitr::purl()`.

The function `mark()` renders Markdown to an output format via the **commonmark** package.

Usage

```
fuse(input, output = NULL, text = NULL, envir = parent.frame(), quiet = FALSE)

fiss(input, output = ".R", text = NULL)

mark(input, output = NULL, text = NULL, options = NULL, meta = list())
```

Arguments

`input`	A character vector to provide the input file path or text. If not provided, the `text` argument must be provided instead. The `input` vector will be treated as a file path if it is a single string, and points to an existing file or has a filename extension. In other cases, the vector will be treated as the `text` argument input. To avoid ambiguity, if a string should be treated as `text` input when it happens to be an existing file path or has an extension, wrap it in `I()`, or simply use the `text` argument instead.

output — An output file path or a filename extension (e.g., `.html`, `.tex`, `.xml`, `.man`, `.markdown`, or `.txt`). In the latter case, the output file path will use the extension on the same base filename as the input file if the `input` is a file. If `output` is not character (e.g., `NA`), the results will be returned as a character vector instead of being written to a file. If `output` is `NULL` or an extension, and the input is a file path, the output file path will have the same base name as the input file, with an extension corresponding to the output format. The output format is retrieved from the first value in the `output` field of the YAML metadata of the `input` (e.g., `html` will generate HTML output). The `output` argument can also take an output format name (possible values are `html`, `latex`, `xml`, `man`, `commonmark`, and `text`). If no output format is detected or provided, the default is HTML.

text — A character vector as the text input. By default, it is read from the `input` file if provided.

envir — An environment in which the code is to be evaluated. It can be accessed via `fuse_env()` inside `fuse()`.

quiet — If `TRUE`, do not show the progress bar. If `FALSE`, the progress bar will be shown after a number of seconds, which can be set via a global option `litedown.progress.delay` (the default is `2`). THe progress bar output can be set via a global option `litedown.progress.output` (the default is `stderr()`).

options — Options to be passed to the renderer. See `markdown_options()` for details. 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)`.

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 Section "YAML metadata" in the documentation for supported variables.

Value

The output file path if output is written to a file, otherwise a character vector of the rendered output (wrapped in `xfun::raw_string()` for clearer printing).

See Also

`sieve()`, for the syntax of R scripts to be passed to `fuse()`.

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

Examples

```
library(litedown)
doc = c("```{r}", "1 + 1", "```", "", "$\\pi$ = `{r} pi`.")
fuse(doc)
fuse(doc, ".tex")
fiss(doc)
```

```
mark(c("Hello _World_!", "", "Welcome to **litedown**."))
# if input appears to be a file path 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")

# output to a file
(mark("_Hello_, **World**!", output = tempfile()))

# convert to other formats
mark("Hello _World_!", ".tex")
mark("Hello _**`World`**_!", "xml")
mark("Hello _**`World`**_!", "text")
```

`fuse_book` *Fuse multiple R Markdown documents to a single output file*

Description

This is a helper function to `fuse()` `.Rmd` files and convert all their Markdown output to a single output file, which is similar to `bookdown::render_book()`, but one major differences is that all HTML output is written to one file, instead of one HTML file per chapter.

Usage

```
fuse_book(input = ".", output = NULL, envir = parent.frame())
```

Arguments

`input`	A directory or a vector of file paths. By default, all `.Rmd`/`.md` files under the current working directory are used as the input, except for filenames that start with `.` or `_` (e.g., `_foo.Rmd`), or `.md` files with the same base names as `.Rmd` files (e.g., `bar.md` will not be used if `bar.Rmd` exists). For a directory `input`, the file search will be recursive if `input` ends with a slash (i.e., sub-directories will also be searched). If a file named `index.Rmd` or `index.md` exists, it will always be treated as the first input file. Input files can also be specified in the config file `_litedown.yml` (in the `input` field under `book`).
`output`	An output file path or a filename extension (e.g., `.html`, `.tex`, `.xml`, `.man`, `.markdown`, or `.txt`). In the latter case, the output file path will use the extension on the same base filename as the input file if the `input` is a file. If `output` is not character (e.g., `NA`), the results will be returned as a character vector instead of being written to a file. If `output` is `NULL` or an extension, and the input is a file path, the output file path will have the same base name as the input file, with an extension corresponding to the output format. The output format is retrieved from the first value in the `output` field of the YAML metadata of the `input` (e.g., `html` will generate HTML output). The `output` argument can also take an output format name (possible values are `html`, `latex`, `xml`, `man`, `commonmark`, and `text`). If no output format is detected or provided, the default is HTML.

envir An environment in which the code is to be evaluated. It can be accessed via fuse_env() inside fuse().

Details

If the output format needs to be customized, the settings should be written in the config file _litedown.yml, e.g.,

```
---
output:
  html:
    options:
      toc:
        depth: 4
  latex:
    meta:
      documentclass: "book"
```

In addition, you can configure the book via the book field, e.g.,

```
---
book:
  new_session: true
  subdir: false
  pattern: "[.]R?md$"
  chapter_before: "Information before a chapter."
  chapter_after: "This chapter was generated from `$input$`."
---
```

The option new_session specifies whether to render each input file in the current R session or a separate new R session; chapter_before and chapter_after specify text to be added to the beginning and end of each file, respectively, which accepts some variables (e.g., $input$ is the current input file path).

Value

An output file path or the output content, depending on the output argument.

---

fuse_env *The* fuse() *environment*

Description

Get the environment passed to the envir argument of fuse(), i.e., the environment in which code chunks and inline code are evaluated.

Usage

```
fuse_env()
```

Value

When called during `fuse()`, it returns the `envir` argument value of `fuse()`. When called outside `fuse()`, it returns the global environment.

`fuse_site` *Fuse R Markdown documents individually under a directory*

Description

Run `fuse()` on R Markdown documents individually to generate a website.

Usage

```
fuse_site(input = ".")
```

Arguments

`input`	The root directory of the site, or a vector of input file paths.

Details

If a directory contains a config file `_litedown.yml`, which has a YAML field `site`, the directory will be recognized as a site root directory. The YAML field `output` will be applied to all R Markdown files (an individual R Markdown file can provide its own `output` field in YAML to override the global config). For example:

```
---
site:
  rebuild: "outdated"
  pattern: "[.]R?md$"
output:
  html:
    meta:
      css: ["@default"]
      include_before: "[Home](/) [About](/about.html)"
      include_after: "&copy; 2024 | [Edit]($input$)"
---
```

The option `rebuild` determines whether to rebuild `.Rmd` files. Possible values are:

- `newfile`: Build an input file if it does not have a `.html` output file.
- `outdated`: Rebuild an input file if the modification time of its `.html` output file is older than the input.

Value

Output file paths (invisibly).

---

get_context *Get the* fuse() *context*

Description

A helper function to query the fuse() context (such as the input file path or the output format name) when called inside a code chunk.

Usage

```
get_context(item = NULL)
```

Arguments

item	The name of the context item.

Value

If the item is provided, return its value in the context. If NULL, the whole context (an environment) is returned.

Examples

```
litedown::get_context("input")
litedown::get_context("format")
names(litedown::get_context())  # all available items
```

---

html_format *Output formats in YAML metadata*

Description

These functions exist only for historical reasons, and should never be called directly. They can be used to configure output formats in YAML, but you are recommended to use the file format names instead of these function names.

Usage

```
html_format(options = NULL, meta = NULL, template = NULL, keep_md = FALSE)

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

Arguments

meta, options	Arguments to be passed to mark().
template	A template file path.
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'.
citation_package	The LaTeX package for processing citations. Possible values are none, natbib, and biblatex.

Details

To configure output formats in the YAML metadata of the Markdown document, simply use the output format names such as html or latex in the output field in YAML, e.g.,

```
---
output:
  html:
    options:
      toc: true
    keep_md: true
  latex:
    latex_engine: pdflatex
---
```

You can also use litedown::html_format instead of html (or litedown::latex_format instead of latex) if you like.

Note

If you want to use the Knit button in RStudio, you must add a top-level field knit: litedown:::knit to the YAML metadata. See https://yihui.org/litedown/#sec:knit-button for more information.

`markdown_options` *Markdown rendering options*

Description

A list of all 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

See https://yihui.org/litedown/#sec:markdown-options for the full list of options and their documentation.

Value

A character vector of all available options.

Examples

```
# all available options
litedown::markdown_options()

library(litedown)

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

mark(mkd, options = "+number_sections")
mark(mkd, options = "+number_sections+toc")

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

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

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

```
# table example
mark("
First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
")

# caption
mark("
| a | b |
|---|--:|
| A | 9 |

Table: A table _caption_.
")

# no table
mark("
First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
", options = '-table')

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

# links and spans
mark('[a b](#){.red}')
mark('[a\nb](){.red}')

# strikethrough example
mark("~~awesome~~")
mark("~~awesome~~", options = "-strikethrough")

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

# code blocks
mark('```r\n1 + 1;\n```')
mark('```{.r}\n1 + 1;\n```')
mark('```{.r .js}\n1 + 1;\n```')
mark('```{.r .js #foo}\n1 + 1;\n```')
mark('```{.r .js #foo style="background:lime;"}\n1 + 1;\n```')
mark('````\nA _code chunk_:\n\n```{r, echo=TRUE}\n1 + 1;\n```\n````')

# raw blocks
mark('```{=html}\n<p>raw HTML</p>\n```')
```

```
mark('```{=latex}\n\\textbf{raw LaTeX}\n```')

# filter out HTML tags
mkd = '<style>a {}</style><script type="text/javascript">console.log("No!");</script>\n[Hello](#)'
mark(mkd)
# tagfiler doesn't work: https://github.com/r-lib/commonmark/issues/15
# mark(mkd, options = "tagfilter")
```

---

pkg_desc *Print the package description, news, citation, manual pages, and source code*

Description

Helper functions to retrieve various types of package information that can be put together as the full package documentation like a **pkgdown** website. These functions can be called inside any R Markdown document.

Usage

```
pkg_desc(name = detect_pkg())

pkg_news(
  name = detect_pkg(),
  path = detect_news(name),
  recent = 1,
  toc = TRUE,
  number_sections = TRUE,
  ...
)

pkg_code(
  path = attr(detect_pkg(), "path"),
  pattern = "[.](R|c|h|f|cpp)$",
  toc = TRUE,
  number_sections = TRUE,
  link = TRUE
)

pkg_citation(name = detect_pkg())

pkg_manual(
  name = detect_pkg(),
  toc = TRUE,
  number_sections = TRUE,
  overview = TRUE,
  examples = list()
)
```

Arguments

name	The package name (by default, it is automatically detected from the DESCRIPTION file if it exists in the current working directory or upper-level directories).
path	For pkg_news(), path to the NEWS.md file. If empty, news() will be called to retrieve the news entries. For pkg_code(), path to the package root directory that contains R/ and/or src/ subdirectories.
recent	The number of recent versions to show. By default, only the latest version's news entries are retrieved. To show the full news, set recent = 0.
toc	Whether to add section headings to the document TOC (when TOC has been enabled for the document).
number_sections	Whether to number section headings (when sections are numbered in the document).
...	Other arguments to be passed to news().
pattern	A regular expression to match filenames that should be treated as source code.
link	Whether to add links on the file paths of source code. By default, if a GitHub repo link is detected from the BugReports field of the package DESCRIPTION, GitHub links will be added to file paths. You can also provide a string template containing the placeholder %s (which will be filled out with the file paths via sprintf()), e.g., https://github.com/yihui/litedown/blob/main/%s.
overview	Whether to include the package overview page, i.e., the {name}-package.Rd page.
examples	A list of arguments to be passed to xfun::record() to run examples each help page, e.g., list(dev = 'svg', dev.args = list(height = 6)). If not a list (e.g., FALSE), examples will not be run.

Value

A character vector (HTML or Markdown) that will be printed as is inside a code chunk of an R Markdown document.

pkg_desc() returns an HTML table containing the package metadata.

pkg_news() returns the news entries.

pkg_code() returns the package source code under the R/ and src/ directories.

pkg_citation() returns the package citation in both the plain-text and BibTeX formats.

pkg_manual() returns all manual pages of the package in HTML.

Examples

```
## Not run:
litedown::pkg_desc()
litedown::pkg_news()
litedown::pkg_citation()

## End(Not run)
```

raw_text *Mark a character vector as raw output*

Description

This function should be called inside a code chunk, and its effect is the same as the chunk option `results = "asis"`. The input character vector will be written verbatim to the output (and interpreted as Markdown).

Usage

```
raw_text(x, format = NULL)
```

Arguments

`x`	A character vector (each element will be treated as a line).
`format`	An output format name, e.g., `html` or `latex`. If provided, `x` will be wrapped in a fenced code block, e.g., ```` ```{=html} ````.

Value

A character vector with a special class to indicate that it should be treated as raw output.

Examples

```
litedown::raw_text(c("**This**", "_is_", "[Markdown](#)."))
litedown::raw_text("<b>Bold</b>", "html")
litedown::raw_text("\\textbf{Bold}", "latex")
```

reactor *Get and set chunk options*

Description

Chunk options are stored in an environment returned by `reactor()`. Option values can be queried by passing their names to `reactor()`, and set by passing named values.

Usage

```
reactor(...)
```

Arguments

`...`	Named values (for setting) or unnamed values (for getting).

Value

With no arguments, `reactor()` returns an environment that stores the options, which can also be used to get or set options. For example, with `opts = reactor()`, `opts$name` returns an option value, and `opts$name = value` sets an option to a value.

With named arguments, `reactor()` sets options and returns a list of their old values (e.g., `reactor(echo = FALSE, fig.width = 8)`). The returned list can be passed to `reactor()` later to restore the options.

With unnamed arguments, `reactor()` returns option values after received option names as input. If one name is received, its value is returned (e.g., `reactor('echo')`). If multiple names are received, a named list of values is returned (e.g., `reactor(c('echo', 'fig.width'))`). A special case is that if only one unnamed argument is received and it takes a list of named values, the list will be used to set options, e.g., `reactor(list(echo = FALSE, fig.width = 8))`, which is equivalent to `reactor(echo = FALSE, fig.width = 8)`.

Examples

```
# get options
litedown::reactor("echo")
litedown::reactor(c("echo", "fig.width"))

# set options
old = litedown::reactor(echo = FALSE, fig.width = 8)
litedown::reactor(c("echo", "fig.width"))
litedown::reactor(old)  # restore options

# use the environment directly
opts = litedown::reactor()
opts$echo
mget(c("echo", "fig.width"), opts)
ls(opts)  # built-in options
```

`roam` *Preview Markdown and R Markdown files*

Description

Launch a web page to list and preview files under a directory.

Usage

```
roam(dir = ".", live = TRUE, ...)
```

Arguments

`dir`	A directory path.

live	Whether to enable live preview. If enabled, the browser page will be automatically updated upon modification of local files used by the page (e.g., the Markdown file or external CSS/JS/image files). If disabled, you can manually refresh the page to fully re-render it.
...	Other arguments to be passed to xfun::new_app().

Details

Markdown files will be converted to HTML and returned to the web browser directly without writing to HTML files, to keep the directory clean during the preview. Clicking on a filename will bring up an HTML preview. To see its raw content, click on the link on its file size instead.

Value

A URL (invisibly) for the preview.

---

timing_data *Get the timing data of code chunks and text blocks in a document*

Description

Timing can be enabled via the chunk option time = TRUE (e.g., set reactor(time = TRUE) in the first code chunk). After it is enabled, the execution time for code chunks and text blocks will be recorded. This function can be called to retrieve the timing data later in the document (e.g., in the last code chunk).

Usage

```
timing_data(threshold = 0, sort = TRUE, total = TRUE)
```

Arguments

threshold	A number (time in seconds) to subset data with. Only rows with time above this threshold are returned.
sort	Whether to sort the data by time in the decreasing order.
total	Whether to append the total time to the data.

Value

A data frame containing input file paths, line numbers, chunk labels, and time. If no timing data is available, NULL is returned.

Note

By default, the data will be cleared after each call of fuse() and will not be available outside fuse(). To store the data persistently, you can set the time option to a file path. This is necessary if you want to get the timing data for multiple input documents (such as all chapters of a book). Each document needs to point the time option to the same path. When you do not need timing any more, you will need to delete this file by yourself.

vest *Add CSS/JS assets to HTML output*

Description

While CSS/JS assets can be set via the `css`/`js` keys under the `meta` field of the `html` output format in YAML, this function provides another way to add them, which can be called in a code chunk to dynamically add assets.

Usage

```
vest(feature = NULL, css = NULL, js = NULL)
```

Arguments

`feature`	A character vector of features supported by CSS/JS, e.g., `c('article', 'callout')`. See the row names of `litedown:::assets` for all available features. Each feature will be mapped to CSS/JS.
`css`, `js`	Character vectors of CSS/JS assets.

Value

A vector of `<link>` (CSS) or `<script>` (JS) tags.

Examples

```
litedown:::assets[, -1]
# add features
litedown::vest(c("copy-button", "tabsets"))
# add css/js directly
litedown::vest(css = "@tabsets", js = c("@tabsets", "@fold-details"))
```

Index