Package 'tinkr'

May 8, 2026

Title Cast '(R)Markdown' Files to 'XML' and Back Again

Version 0.3.1

Description Parsing '(R)Markdown' files with numerous regular expressions can be fraught with peril, but it does not have to be this way. Converting '(R)Markdown' files to 'XML' using the 'commonmark' package allows in-memory editing via of 'markdown' elements via 'XPath' through the extensible 'R6' class called 'yarn'. These modified 'XML' representations can be written to '(R)Markdown' documents via an 'xslt' stylesheet which implements an extended version of 'GitHub'-flavoured 'markdown' so that you can tinker to your hearts content.

License GPL-3

URL https://docs.ropensci.org/tinkr/, https://github.com/ropensci/tinkr

BugReports https://github.com/ropensci/tinkr/issues

Imports commonmark (>= 1.6), glue, lifecycle, magrittr, purrr, R6, rlang (>= 0.4.5), xml2, xslt

Suggests knitr, rmarkdown, covr, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2.9000

VignetteBuilder knitr

Config/Needs/build moodymudskipper/devtag

Language en-US

NeedsCompilation no

Author Maëlle Salmon [aut] (ORCID: <https://orcid.org/0000-0002-2815-0399>), Zhian N. Kamvar [aut, cre] (ORCID: <https://orcid.org/0000-0003-1458-7108>), Jeroen Ooms [aut], Nick Wellnhofer [cph] (Nick Wellnhofer wrote the XSLT stylesheet.), rOpenSci [fnd] (ROR: <https://ror.org/019jywm96>), Peter Daengeli [ctb]

Maintainer Zhian N. Kamvar <zkamvar@gmail.com>

Depends R (>= 4.1.0)

Repository CRAN

Date/Publication 2025-10-04 19:50:03 UTC

Contents

`find_between` *Find between a pattern*

Description

Helper function to find all nodes between a standard pattern. This is useful if you want to find unnested pandoc tags.

Usage

```
find_between(
  body,
  ns,
  pattern = "md:paragraph[md:text[starts-with(text(), ':::')]]",
  include = FALSE
)
```

Arguments

`body`	and XML document
`ns`	the namespace of the document
`pattern`	an XPath expression that defines characteristics of nodes between which you want to extract everything.
`include`	if `TRUE`, the tags matching `pattern` will be included in the output, defaults to `FALSE`, which only gives you the nodes in between `pattern`.

Value

a nodeset

Examples

```
md <- glue::glue("
 h1
 ====

 ::: section

 h2
 ----

 section *text* with [a link](https://ropensci.org/)

 :::
")
x <- xml2::read_xml(commonmark::markdown_xml(md))
ns <- xml2::xml_ns_rename(xml2::xml_ns(x), d1 = "md")
res <- find_between(x, ns)
res
xml2::xml_text(res)
xml2::xml_find_all(res, ".//descendant-or-self::md:*", ns = ns)
```

`get_protected` *Get protected nodes*

Description

Get protected nodes

Usage

```
get_protected(body, type = NULL, ns = md_ns())
```

Arguments

`body`	an `xml_document` object
`type`	a character vector listing the protections to be included. Defaults to `NULL`, which includes all protected nodes: • math: via the `protect_math()` function • curly: via the `protect_curly()` function • unescaped: via the `protect_unescaped()` function
`ns`	the namespace of the document (defaults to `md_ns()`)

Value

an xml_nodelist object.

Examples

```
path <- system.file("extdata", "basic-curly.md", package = "tinkr")
ex <- tinkr::yarn$new(path, sourcepos = TRUE)
# protect curly braces
ex$protect_curly()
# add fenced divs and protect then
ex$add_md(c("::: alert\n",
  "blabla",
  ":::")
)
ex$protect_fences()
# add math and protect it
ex$add_md(c("## math\n",
  "$c^2 = a^2 + b^2$\n",
  "$$",
  "\\sum_{i}^k = x_i + 1",
  "$$\n")
)
ex$protect_math()
# get protected now shows all the protected nodes
get_protected(ex$body)
get_protected(ex$body, c("math", "curly")) # only show the math and curly
```

---

md_ns *Aliased namespace prefix for commonmark*

Description

The commonmark package is used to translate markdown to XML, but it does not assign a namespace prefix, which means that xml2 will auto-assign a default prefix of d1.

Usage

```
md_ns()
```

Details

This function renames the default prefix to md, so that you can use XPath queries that are slightly more descriptive.

Value

an xml_namespace object (see xml2::xml_ns())

Examples

```
tink <- tinkr::to_xml(system.file("extdata", "example1.md", package = "tinkr"))
# with default namespace
xml2::xml_find_all(tink$body,
  ".//d1:link[starts-with(@destination, 'https://ropensci')]"
)
# with tinkr namespace
xml2::xml_find_all(tink$body,
  ".//md:link[starts-with(@destination, 'https://ropensci')]",
  tinkr::md_ns()
)
```

---

protect_curly *Protect curly elements for further processing*

Description

Protect curly elements for further processing

Usage

```
protect_curly(body, ns = md_ns())
```

Arguments

body	an XML object
ns	an XML namespace object (defaults: `md_ns()`).

Details

Commonmark will render text such as `{.unnumbered}` (Pandoc/Quarto option) or `{#hello .greeting .message style="c` (Markdown custom block) as normal text which might be problematic if trying to extract real text from the XML.

If sending the XML to, say, a translation API that allows some tags to be ignored, you could first transform the text tags with the attribute `curly` to `curly` tags, and then transform them back to text tags before using `to_md()`.

Value

a copy of the modified XML object

Note

this function is also a method in the yarn object.

Examples

```
m <- tinkr::to_xml(system.file("extdata", "basic-curly.md", package = "tinkr"))
xml2::xml_child(m$body)
m$body <- protect_curly(m$body)
xml2::xml_child(m$body)
```

`protect_fences` *Protect fences of Pandoc fences divs for further processing*

Description

Protect fences of Pandoc fences divs for further processing

Usage

```
protect_fences(body, ns = md_ns())
```

Arguments

`body`	an XML object
`ns`	an XML namespace object (defaults: `md_ns()`).

Details

Commonmark will render text such as `::: footer` as normal text which might be problematic if trying to extract real text from the XML.

If sending the XML to, say, a translation API that allows some tags to be ignored, you could first transform the text tags with the attribute `fences` to `fences` tags, and then transform them back to text tags before using `to_md()`.

Value

a copy of the modified XML object

Note

this function is also a method in the yarn object.

Examples

```
m <- tinkr::to_xml(system.file("extdata", "fenced-divs.md", package = "tinkr"))
xml2::xml_child(m$body)
m$body <- protect_fences(m$body)
xml2::xml_child(m$body)
```

protect_math *Protect math elements from commonmark's character escape*

Description

Protect math elements from commonmark's character escape

Usage

```
protect_math(body, ns = md_ns())
```

Arguments

body	an XML object
ns	an XML namespace object (defaults: md_ns()).

Details

Commonmark does not know what LaTeX is and will LaTeX equations as normal text. This means that content surrounded by underscores are interpreted as <emph> elements and all backslashes are escaped by default. This function protects inline and block math elements that use $ and $$ for delimiters, respectively.

Value

a copy of the modified XML object

Note

this function is also a method in the yarn object.

Examples

```
m <- tinkr::to_xml(system.file("extdata", "math-example.md", package = "tinkr"))
txt <- textConnection(tinkr::to_md(m))
cat(tail(readLines(txt)), sep = "\n") # broken math
close(txt)
m$body <- protect_math(m$body)
txt <- textConnection(tinkr::to_md(m))
cat(tail(readLines(txt)), sep = "\n") # fixed math
close(txt)
```

show_list *Display a node or nodelist as markdown*

Description

When inspecting the results of an XPath query, displaying the text often

Usage

```
show_list(nodelist, stylesheet_path = stylesheet())

show_block(nodelist, mark = FALSE, stylesheet_path = stylesheet())

show_censor(nodelist, stylesheet_path = stylesheet())
```

Arguments

nodelist	an object of class xml_nodeset OR xml_node OR a list of either.
stylesheet_path	path to the XSL stylesheet
mark	[bool] When TRUE markers ([...]) are added to replace nodes that come before or after the islated nodes. Defaults to FALSE, which only shows the isolated nodes in their respective blocks. Note that the default state may cause nodes within the same block to appear adjacent to each other.

Value

a character vector, invisibly. The result of these functions are displayed to the screen

See Also

to_md_vec() to get a vector of these elements in isolation.

Examples

```
path <- system.file("extdata", "show-example.md", package = "tinkr")
y <- tinkr::yarn$new(path, sourcepos = TRUE)
y$protect_math()$protect_curly()
items <- xml2::xml_find_all(y$body, ".//md:item", tinkr::md_ns())
imgs <- xml2::xml_find_all(y$body, ".//md:image | .//node()[@curly]",
  tinkr::md_ns())
links <- xml2::xml_find_all(y$body, ".//md:link", tinkr::md_ns())
code <- xml2::xml_find_all(y$body, ".//md:code", tinkr::md_ns())
blocks <- xml2::xml_find_all(y$body, ".//md:code_block", tinkr::md_ns())

# show a list of items
show_list(links)
show_list(code)
```

```
show_list(blocks)

# show the items in their local structure
show_block(items)
show_block(links, mark = TRUE)

# show the items in the full document censored (everything but whitespace):
show_censor(imgs)

# You can also adjust the censorship parameters. There are two paramters
# available: the mark, which chooses what character you want to use to
# replace characters (default: `\u2587`); and the regex which specifies
# characters to replace (default: `[^[:space:]]`, which replaces all
# non-whitespace characters.
#
# The following will replace everything that is not a whitespace
# or punctuation character with "o" for a very ghostly document
op <- options()
options(tinkr.censor.regex = "[^[:space:][:punct:]]")
options(tinkr.censor.mark = "o")
show_censor(links)
options(tinkr.censor.regex = NULL)
options(tinkr.censor.mark = NULL)
```

`stylesheet` *The tinkr stylesheet*

Description

This function returns the path to the tinkr stylesheet

Usage

```
stylesheet()
```

Value

a single element character vector representing the path to the stylesheet used by tinkr.

Examples

```
tinkr::stylesheet()
```

to_md *Write front-matter (YAML, TOML or JSON) and XML back to disk as (R)Markdown*

Description

Write front-matter (YAML, TOML or JSON) and XML back to disk as (R)Markdown

Usage

```
to_md(
  frontmatter_xml_list,
  path = NULL,
  stylesheet_path = stylesheet(),
  yaml_xml_list = deprecated()
)

to_md_vec(nodelist, stylesheet_path = stylesheet())
```

Arguments

frontmatter_xml_list	result from a call to `to_xml()` and editing.
path	path of the new file. Defaults to `NULL`, which will not write any file, but will still produce the conversion and pass the output as a character vector.
stylesheet_path	path to the XSL stylesheet
yaml_xml_list	**[Deprecated]** Use `frontmatter_xml_list()`.
nodelist	an object of `xml_nodelist` or `xml_node`

Details

The stylesheet you use will decide whether lists are built using "*" or "-" for instance. If you're keen to keep your own Markdown style when using `to_md()` after `to_xml()`, you can tweak the XSL stylesheet a bit and provide the path to your XSL stylesheet as argument.

Value

- `to_md()`: `\[character\]` the converted document, invisibly as a character vector containing two elements: the frontmatter list and the markdown body.
- `to_md_vec()`: `\[character\]` the markdown representation of each node.

Examples

```
path <- system.file("extdata", "example1.md", package = "tinkr")
frontmatter_xml_list <- to_xml(path)
names(frontmatter_xml_list)
# extract the level 3 headers from the body
headers3 <- xml2::xml_find_all(
  frontmatter_xml_list$body,
  xpath = './/md:heading[@level="3"]',
  ns = md_ns()
)
# show the headers
print(h3 <- to_md_vec(headers3))
# transform level 3 headers into level 1 headers
# NOTE: these nodes are still associated with the document and this is done
# in place.
xml2::xml_set_attr(headers3, "level", 1)
# preview the new headers
print(h1 <- to_md_vec(headers3))
# save back and have a look
newmd <- tempfile("newmd", fileext = ".md")
res <- to_md(frontmatter_xml_list, newmd)
# show that it works
regmatches(res[[2]], gregexpr(h1[1], res[[2]], fixed = TRUE))
# file.edit("newmd.md")
file.remove(newmd)
```

---

to_xml *Transform file to XML*

Description

Transform file to XML

Usage

```
to_xml(
  path,
  encoding = "UTF-8",
  sourcepos = FALSE,
  anchor_links = TRUE,
  unescaped = TRUE
)
```

Arguments

path	Path to the file.
encoding	Encoding to be used by readLines.

sourcepos	passed to commonmark::markdown_xml(). If TRUE, the source position of the file will be included as a "sourcepos" attribute. Defaults to FALSE.
anchor_links	if TRUE (default), reference-style links with anchors (in the style of [key]: https://example.com/link will be preserved as best as possible. If this is FALSE, the anchors disappear and the links will appear as normal links. See resolve_anchor_links() for details.
unescaped	if TRUE (default) AND sourcepos = TRUE, square braces that were unescaped in the original document will be preserved as best as possible. If this is FALSE, these braces will be escaped in the output document. See protect_unescaped() for details.

Details

This function will take a (R)markdown file, split the frontmatter from the body, and read in the body through commonmark::markdown_xml(). Any RMarkdown code fences will be parsed to expose the chunk options in XML and tickboxes (aka checkboxes) in GitHub-flavored markdown will be preserved (both modifications from the commonmark standard).

Value

A list containing the front-matter (YAML, TOML or JSON) of the file (frontmatter) and its body (body) as XML.

Note

Math elements are not protected by default. You can use protect_math() to address this if needed.

Examples

```
path <- system.file("extdata", "example1.md", package = "tinkr")
post_list <- to_xml(path)
names(post_list)
path2 <- system.file("extdata", "example2.Rmd", package = "tinkr")
post_list2 <- to_xml(path2)
post_list2
```

---

yarn *R6 class containing XML representation of Markdown*

Description

Wrapper around an XML representation of a Markdown document. It contains four publicly accessible slots: path, frontmatter, body, and ns.

Details

This class is a fancy wrapper around the results of to_xml() and has methods that make it easier to add, analyze, remove, or write elements of your markdown document.

Public fields

`path` [character] path to file on disk

`frontmatter` [character] text block at head of file

`frontmatter_format` [character] 'YAML', 'TOML' or 'JSON'

`body` [xml_document] an xml document of the (R)Markdown file.

`ns` [xml_document] an xml namespace object defining "md" to commonmark.

Active bindings

`yaml` [character] **[Deprecated]** `frontmatter`

Methods

Public methods:

- `yarn$new()`
- `yarn$reset()`
- `yarn$write()`
- `yarn$show()`
- `yarn$head()`
- `yarn$tail()`
- `yarn$md_vec()`
- `yarn$add_md()`
- `yarn$append_md()`
- `yarn$prepend_md()`
- `yarn$protect_math()`
- `yarn$protect_curly()`
- `yarn$protect_fences()`
- `yarn$protect_unescaped()`
- `yarn$get_protected()`
- `yarn$clone()`

Method `new()`**:** Create a new yarn document

Usage:

```
yarn$new(path = NULL, encoding = "UTF-8", sourcepos = FALSE, ...)
```

Arguments:

`path` [character] path to a markdown episode file on disk

`encoding` [character] encoding passed to `readLines()`

`sourcepos` passed to `commonmark::markdown_xml()`. If `TRUE`, the source position of the file will be included as a "sourcepos" attribute. Defaults to `FALSE`.

`...` arguments passed on to `to_xml()`.

Returns: A new yarn object containing an XML representation of a (R)Markdown file.

Examples:

```
path <- system.file("extdata", "example1.md", package = "tinkr")
ex1 <- tinkr::yarn$new(path)
ex1
path2 <- system.file("extdata", "example2.Rmd", package = "tinkr")
ex2 <- tinkr::yarn$new(path2)
ex2
```

Method `reset()`**:** reset a yarn document from the original file

Usage:

```
yarn$reset()
```

Examples:

```
path <- system.file("extdata", "example1.md", package = "tinkr")
ex1 <- tinkr::yarn$new(path)
# OH NO
ex1$body
ex1$body <- xml2::xml_missing()
ex1$reset()
ex1$body
```

Method `write()`**:** Write a yarn document to Markdown/R Markdown

Usage:

```
yarn$write(path = NULL, stylesheet_path = stylesheet())
```

Arguments:

`path` path to the file you want to write

`stylesheet_path` path to the xsl stylesheet to convert XML to markdown.

Examples:

```
path <- system.file("extdata", "example1.md", package = "tinkr")
ex1 <- tinkr::yarn$new(path)
ex1
tmp <- tempfile()
try(readLines(tmp)) # nothing in the file
ex1$write(tmp)
head(readLines(tmp)) # now a markdown file
unlink(tmp)
```

Method `show()`**:** show the markdown contents on the screen

Usage:

```
yarn$show(lines = TRUE, stylesheet_path = stylesheet())
```

Arguments:

`lines` a subset of elements to show. Defaults to `TRUE`, which shows all lines of the output. This can be either logical or numeric.

`stylesheet_path` path to the xsl stylesheet to convert XML to markdown.

Returns: a character vector with one line for each line in the output

Examples:

```
path <- system.file("extdata", "example2.Rmd", package = "tinkr")
ex2 <- tinkr::yarn$new(path)
ex2$head(5)
ex2$tail(5)
ex2$show()
```

Method `head()`**:** show the head of the markdown contents on the screen

Usage:

```
yarn$head(n = 6L, stylesheet_path = stylesheet())
```

Arguments:

`n` the number of elements to show from the top. Negative numbers

`stylesheet_path` path to the xsl stylesheet to convert XML to markdown.

Returns: a character vector with `n` elements

Method `tail()`**:** show the tail of the markdown contents on the screen

Usage:

```
yarn$tail(n = 6L, stylesheet_path = stylesheet())
```

Arguments:

`n` the number of elements to show from the bottom. Negative numbers

`stylesheet_path` path to the xsl stylesheet to convert XML to markdown.

Returns: a character vector with `n` elements

Method `md_vec()`**:** query and extract markdown elements

Usage:

```
yarn$md_vec(xpath = NULL, stylesheet_path = stylesheet())
```

Arguments:

`xpath` a valid XPath expression

`stylesheet_path` path to the xsl stylesheet to convert XML to markdown.

Returns: a vector of markdown elements generated from the query

Examples:

```
path <- system.file("extdata", "example1.md", package = "tinkr")
ex <- tinkr::yarn$new(path)
# all headings
ex$md_vec(".//md:heading")
# all headings greater than level 3
ex$md_vec(".//md:heading[@level>3]")
# all links
ex$md_vec(".//md:link")
# all links that are part of lists
ex$md_vec(".//md:list//md:link")
# all code
ex$md_vec(".//md:code | .//md:code_block")
```

Method `add_md()`**:** add an arbitrary Markdown element to the document

Usage:

```
yarn$add_md(md, where = 0L)
```

Arguments:

`md` a string of markdown formatted text.

`where` the location in the document to add your markdown text. This is passed on to `xml2::xml_add_child()`. Defaults to 0, which indicates the very top of the document.

Examples:

```
path <- system.file("extdata", "example2.Rmd", package = "tinkr")
ex <- tinkr::yarn$new(path)
# two headings, no lists
xml2::xml_find_all(ex$body, "md:heading", ex$ns)
xml2::xml_find_all(ex$body, "md:list", ex$ns)
ex$add_md(
  "# Hello\n\nThis is *new* formatted text from `{tinkr}`!",
  where = 1L
)$add_md(
  " - This\n - is\n - a new list",
  where = 2L
)
# three headings
xml2::xml_find_all(ex$body, "md:heading", ex$ns)
xml2::xml_find_all(ex$body, "md:list", ex$ns)
tmp <- tempfile()
ex$write(tmp)
readLines(tmp, n = 20)
```

Method `append_md()`**:** append abritrary markdown to a node or set of nodes

Usage:

```
yarn$append_md(md, nodes = NULL, space = TRUE)
```

Arguments:

`md` a string of markdown formatted text.

`nodes` an XPath expression that evaulates to object of class `xml_node` or `xml_nodeset` that are all either inline or block nodes (never both). The XPath expression is passed to `xml2::xml_find_all()`. If you want to append a specific node, you can pass that node to this parameter.

`space` if `TRUE`, inline nodes will have a space inserted before they are appended.

Details: this is similar to the `add_md()` method except that it can do the following:

1. append content after a *specific* node or set of nodes
2. append content to multiple places in the document

Examples:

```
path <- system.file("extdata", "example2.Rmd", package = "tinkr")
ex <- tinkr::yarn$new(path)
# append a note after the first heading

txt <- c("> Hello from *tinkr*!", ">", ">  :heart: R")
ex$append_md(txt, ".//md:heading[1]")$head(20)
```

Method `prepend_md()`**:** prepend arbitrary markdown to a node or set of nodes

Usage:

```
yarn$prepend_md(md, nodes = NULL, space = TRUE)
```

Arguments:

`md` a string of markdown formatted text.

`nodes` an XPath expression that evaulates to object of class `xml_node` or `xml_nodeset` that are all either inline or block nodes (never both). The XPath expression is passed to `xml2::xml_find_all()`. If you want to append a specific node, you can pass that node to this parameter.

`space` if `TRUE`, inline nodes will have a space inserted before they are prepended.

Details: this is similar to the `add_md()` method except that it can do the following:

1. prepend content after a *specific* node or set of nodes
2. prepend content to multiple places in the document

Examples:

```
path <- system.file("extdata", "example2.Rmd", package = "tinkr")
ex <- tinkr::yarn$new(path)

# prepend a table description to the birds table
ex$prepend_md("Table: BIRDS, NERDS", ".//md:table[1]")$tail(20)
```

Method `protect_math()`**:** Protect math blocks from being escaped

Usage:

```
yarn$protect_math()
```

Examples:

```
path <- system.file("extdata", "math-example.md", package = "tinkr")
ex <- tinkr::yarn$new(path)
ex$tail() # math blocks are escaped :(
ex$protect_math()$tail() # math blocks are no longer escaped :)
```

Method `protect_curly()`**:** Protect curly phrases `{likethat}` from being escaped

Usage:

```
yarn$protect_curly()
```

Examples:

```
path <- system.file("extdata", "basic-curly.md", package = "tinkr")
ex <- tinkr::yarn$new(path)
ex$protect_curly()$head()
```

Method `protect_fences()`**:** Protect fences of Pandoc fenced divs `:::`

Usage:

```
yarn$protect_fences()
```

Examples:

```
path <- system.file("extdata", "fenced-divs.md", package = "tinkr")
ex <- tinkr::yarn$new(path)
ex$protect_fences()$head()
```

Method `protect_unescaped()`**:** Protect unescaped square braces from being escaped. This is applied by default when you use `yarn$new(sourcepos = TRUE)`.

Usage:

```
yarn$protect_unescaped()
```

Examples:

```
path <- system.file("extdata", "basic-curly.md", package = "tinkr")
ex <- tinkr::yarn$new(path, sourcepos = TRUE, unescaped = FALSE)
ex$tail()
ex$protect_unescaped()$tail()
```

Method `get_protected()`**:** Return nodes whose contents are protected from being escaped

Usage:

```
yarn$get_protected(type = NULL)
```

Arguments:

`type` a character vector listing the protections to be included. Defaults to `NULL`, which includes all protected nodes:

- math: via the `protect_math()` function
- curly: via the `protect_curly()` function
- fence: via the `protect_fences()` function
- unescaped: via the `protect_unescaped()` function

Examples:

```
path <- system.file("extdata", "basic-curly.md", package = "tinkr")
ex <- tinkr::yarn$new(path, sourcepos = TRUE)
# protect curly braces
ex$protect_curly()
# add fenced divs and protect then
ex$add_md(c("::: alert\n",
  "blabla",
  ":::")
)
ex$protect_fences()
# add math and protect it
ex$add_md(c("## math\n",
  "$c^2 = a^2 + b^2$\n",
  "$$",
  "\\sum_{i}^k = x_i + 1",
  "$$\n")
)
ex$protect_math()
# get protected now shows all the protected nodes
ex$get_protected()
ex$get_protected(c("math", "curly")) # only show the math and curly
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
yarn$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Note

this requires the sourcepos attribute to be recorded when the object is initialised. See protect_unescaped() for details.

See Also

to_md_vec() for a way to generate the same vector from a nodelist without a yarn object

Examples

```
## ------------------------------------------------
## Method `yarn$new`
## ------------------------------------------------

path <- system.file("extdata", "example1.md", package = "tinkr")
ex1 <- tinkr::yarn$new(path)
ex1
path2 <- system.file("extdata", "example2.Rmd", package = "tinkr")
ex2 <- tinkr::yarn$new(path2)
ex2

## ------------------------------------------------
## Method `yarn$reset`
## ------------------------------------------------


path <- system.file("extdata", "example1.md", package = "tinkr")
ex1 <- tinkr::yarn$new(path)
# OH NO
ex1$body
ex1$body <- xml2::xml_missing()
ex1$reset()
ex1$body

## ------------------------------------------------
## Method `yarn$write`
## ------------------------------------------------

path <- system.file("extdata", "example1.md", package = "tinkr")
ex1 <- tinkr::yarn$new(path)
ex1
tmp <- tempfile()
try(readLines(tmp)) # nothing in the file
ex1$write(tmp)
head(readLines(tmp)) # now a markdown file
unlink(tmp)
```

```
## ------------------------------------------------
## Method `yarn$show`
## ------------------------------------------------

path <- system.file("extdata", "example2.Rmd", package = "tinkr")
ex2 <- tinkr::yarn$new(path)
ex2$head(5)
ex2$tail(5)
ex2$show()

## ------------------------------------------------
## Method `yarn$md_vec`
## ------------------------------------------------

path <- system.file("extdata", "example1.md", package = "tinkr")
ex <- tinkr::yarn$new(path)
# all headings
ex$md_vec(".//md:heading")
# all headings greater than level 3
ex$md_vec(".//md:heading[@level>3]")
# all links
ex$md_vec(".//md:link")
# all links that are part of lists
ex$md_vec(".//md:list//md:link")
# all code
ex$md_vec(".//md:code | .//md:code_block")

## ------------------------------------------------
## Method `yarn$add_md`
## ------------------------------------------------

path <- system.file("extdata", "example2.Rmd", package = "tinkr")
ex <- tinkr::yarn$new(path)
# two headings, no lists
xml2::xml_find_all(ex$body, "md:heading", ex$ns)
xml2::xml_find_all(ex$body, "md:list", ex$ns)
ex$add_md(
  "# Hello\n\nThis is *new* formatted text from `{tinkr}`!",
  where = 1L
)$add_md(
  " - This\n - is\n - a new list",
  where = 2L
)
# three headings
xml2::xml_find_all(ex$body, "md:heading", ex$ns)
xml2::xml_find_all(ex$body, "md:list", ex$ns)
tmp <- tempfile()
ex$write(tmp)
readLines(tmp, n = 20)

## ------------------------------------------------
## Method `yarn$append_md`
## ------------------------------------------------
```

```
path <- system.file("extdata", "example2.Rmd", package = "tinkr")
ex <- tinkr::yarn$new(path)
# append a note after the first heading

txt <- c("> Hello from *tinkr*!", ">", ">  :heart: R")
ex$append_md(txt, ".//md:heading[1]")$head(20)

## ------------------------------------------------
## Method `yarn$prepend_md`
## ------------------------------------------------

path <- system.file("extdata", "example2.Rmd", package = "tinkr")
ex <- tinkr::yarn$new(path)

# prepend a table description to the birds table
ex$prepend_md("Table: BIRDS, NERDS", ".//md:table[1]")$tail(20)

## ------------------------------------------------
## Method `yarn$protect_math`
## ------------------------------------------------

path <- system.file("extdata", "math-example.md", package = "tinkr")
ex <- tinkr::yarn$new(path)
ex$tail() # math blocks are escaped :(
ex$protect_math()$tail() # math blocks are no longer escaped :)

## ------------------------------------------------
## Method `yarn$protect_curly`
## ------------------------------------------------

path <- system.file("extdata", "basic-curly.md", package = "tinkr")
ex <- tinkr::yarn$new(path)
ex$protect_curly()$head()

## ------------------------------------------------
## Method `yarn$protect_fences`
## ------------------------------------------------

path <- system.file("extdata", "fenced-divs.md", package = "tinkr")
ex <- tinkr::yarn$new(path)
ex$protect_fences()$head()

## ------------------------------------------------
## Method `yarn$protect_unescaped`
## ------------------------------------------------

path <- system.file("extdata", "basic-curly.md", package = "tinkr")
ex <- tinkr::yarn$new(path, sourcepos = TRUE, unescaped = FALSE)
ex$tail()
ex$protect_unescaped()$tail()

## ------------------------------------------------
```

```
## Method `yarn$get_protected`
## ------------------------------------------------

path <- system.file("extdata", "basic-curly.md", package = "tinkr")
ex <- tinkr::yarn$new(path, sourcepos = TRUE)
# protect curly braces
ex$protect_curly()
# add fenced divs and protect then
ex$add_md(c("::: alert\n",
  "blabla",
  ":::")
)
ex$protect_fences()
# add math and protect it
ex$add_md(c("## math\n",
  "$c^2 = a^2 + b^2$\n",
  "$$",
  "\\sum_{i}^k = x_i + 1",
  "$$\n")
)
ex$protect_math()
# get protected now shows all the protected nodes
ex$get_protected()
ex$get_protected(c("math", "curly")) # only show the math and curly
```

Index