Package 'teal.reporter'

February 20, 2026

Title Reporting Tools for 'shiny' Modules

Version 0.6.1

Date 2026-02-16

Description Prebuilt 'shiny' modules containing tools for the generation of 'rmarkdown' reports, supporting reproducible research and analysis.

License Apache License 2.0

URL https://github.com/insightsengineering/teal.reporter, https://insightsengineering.github.io/teal.reporter/

BugReports https://github.com/insightsengineering/teal.reporter/issues

Imports bsicons, bslib (>= 0.8.0), checkmate (>= 2.1.0), commonmark (>= 1.9.2), flextable (>= 0.9.2), grid, gtsummary (>= 2.0.0), htmltools (>= 0.5.4), jsonlite (>= 1.8.9), knitr (>= 1.42), lifecycle (>= 0.2.0), methods, R6, rlang (>= 1.0.0), rlistings (>= 0.2.10), rmarkdown (>= 2.23), rtables (>= 0.6.11), rtables.officer (>= 0.0.2), shiny (>= 1.8.1), shinybusy (>= 0.3.2), shinyjs (>= 2.1.0), shinyWidgets (>= 0.5.1), sortable (>= 0.5.0), teal.code (>= 0.7.0), teal.data (>= 0.8.0), tools, utils, yaml (>= 1.1.0), zip (>= 1.1.0)

Suggests DT (>= 0.13), formatR (>= 1.5), formatters (>= 0.5.10), ggplot2 (>= 3.4.3), lattice (>= 0.18-4), png, shinytest2, testthat (>= 3.2.2), tinytex, waldo (>= 0.2.0), withr (>= 2.0.0)

VignetteBuilder knitr, rmarkdown

Config/Needs/verdepcheck rstudio/bsicons,rstudio/bslib, mllg/checkmate, r-lib/commonmark, davidgohel/flextable, ddsjoberg/gtsummary, rstudio/htmltools, jeroen/jsonlite, yihui/knitr, r-lib/lifecycle, r-lib/R6, r-lib/rlang, insightsengineering/rlistings, rstudio/rmarkdown, insightsengineering/rtables, insightsengineering/rtables.officer, rstudio/shiny, dreamRs/shinybusy, daattali/shinyjs, dreamRs/shinyWidgets, rstudio/sortable, insightsengineering/teal.code,

insightsengineering/teal.data, yaml=vubiostat/r-yaml,
r-lib/zip, rstudio/DT, yihui/formatR,
insightsengineering/formatters, tidyverse/ggplot2,
deepayan/lattice, cran/png, rstudio/shinytest2, r-lib/testthat,
rstudio/tinytex, r-lib/waldo, r-lib/withr

Config/Needs/website insightsengineering/nesttemplate

Config/testthat/edition 3

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

NeedsCompilation no

Author Dawid Kaledkowski [aut, cre] (ORCID:
<https://orcid.org/0000-0001-9533-457X>),
Kartikeya Kirar [aut] (ORCID: <https://orcid.org/0009-0005-1258-4757>),
Marcin Kosinski [aut],
Maciej Nasinski [aut],
Konrad Pagacz [aut],
Mahmoud Hallal [aut],
Chendi Liao [rev],
Dony Unardi [rev],
F. Hoffmann-La Roche AG [cph, fnd]

Maintainer Dawid Kaledkowski <dawid.kaledkowski@roche.com>

Repository CRAN

Date/Publication 2026-02-20 10:20:08 UTC

Contents

`add_card_button` *Add card button module*

Description

Provides a button to add views/cards to a report.

For more details see the vignette: `vignette("simpleReporter", "teal.reporter")`.

Usage

```
add_card_button_ui(id, label = NULL)

add_card_button_srv(id, reporter, card_fun, card_title = "")
```

Arguments

`id`	(`character(1)`) this `shiny` module's id.
`label`	(`character(1)`) label of the button. By default it is empty.
`reporter`	(`Reporter`) instance.
`card_fun`	(`function`) which returns a `ReportCard` instance. See `Details`.
`card_title`	(`character(1)`) default value for the card title input field. By default it is empty.

Details

The `card_fun` function is designed to create a new `ReportCard` instance and optionally customize it:

- The `teal_card` parameter allows for specifying a custom or default `ReportCard` instance.
- Use the `comment` parameter to add a comment to the card via `card$append_text()` - if `card_fun` does not have the `comment` parameter, then `comment` from `Add Card UI` module will be added at the end of the content of the card.
- The `label` parameter enables customization of the card's name and its content through `card$append_text()`- if `card_fun` does not have the `label` parameter, then card name will be set to the name passed in `Add Card UI` module, but no text will be added to the content of the `teal_card`.

This module supports using a subclass of `ReportCard` for added flexibility. A subclass instance should be passed as the default value of the `teal_card` argument in the `card_fun` function. See below:

```
CustomReportCard <- R6::R6Class(
  classname = "CustomReportCard",
  inherit = teal.reporter::ReportCard
)

custom_function <- function(card = CustomReportCard$new()) {
  card
}
```

Value

NULL.

---

as_yaml_auto *Parse a named list to* yaml *header for an* Rmd *file*

Description

Converts a named list into a yaml header for Rmd, handling output types and arguments as defined in the rmarkdown package. This function simplifies the process of generating yaml headers.

Usage

```
as_yaml_auto(
  input_list,
  as_header = TRUE,
  convert_logi = TRUE,
  multi_output = FALSE,
  silent = FALSE
)
```

Arguments

input_list	(named list) non nested with slots names and their values compatible with Rmd yaml header.
as_header	(logical(1)) optionally wrap with result with the internal md_header(), default TRUE.
convert_logi	(logical(1)) convert a character values to logical, if they are recognized as quoted yaml logical values , default TRUE.
multi_output	(logical(1)) multi output slots in the input argument, default FALSE.
silent	(logical(1)) suppress messages and warnings, default FALSE.

Details

This function processes a non-nested (flat) named list into a yaml header for an Rmd document. It supports all standard Rmd yaml header fields, including author, date, title, subtitle, abstract, keywords, subject, description, category, and lang. Additionally, it handles output field types and arguments as defined in the rmarkdown package.

Value

`character` with `rmd_yaml_header` class, result of `yaml::as.yaml`, optionally wrapped with internal `md_header()`.

Note

Only non-nested lists are automatically parsed. Nested lists require direct processing with `yaml::as.yaml`.

Examples

```
# nested so using yaml::as.yaml directly
as_yaml_auto(
  list(author = "", output = list(pdf_document = list(toc = TRUE)))
)

# auto parsing for a flat list, like shiny input
input <- list(author = "", output = "pdf_document", toc = TRUE, keep_tex = TRUE)
as_yaml_auto(input)

as_yaml_auto(list(author = "", output = "pdf_document", toc = TRUE, keep_tex = "TRUE"))

as_yaml_auto(list(
  author = "", output = "pdf_document", toc = TRUE, keep_tex = TRUE,
  wrong = 2
))

as_yaml_auto(list(author = "", output = "pdf_document", toc = TRUE, keep_tex = 2),
  silent = TRUE
)

input <- list(author = "", output = "pdf_document", toc = TRUE, keep_tex = "True")
as_yaml_auto(input)
as_yaml_auto(input, convert_logi = TRUE, silent = TRUE)
as_yaml_auto(input, silent = TRUE)
as_yaml_auto(input, convert_logi = FALSE, silent = TRUE)

as_yaml_auto(
  list(
    author = "", output = "pdf_document",
    output = "html_document", toc = TRUE, keep_tex = TRUE
  ),
  multi_output = TRUE
)
as_yaml_auto(
  list(
    author = "", output = "pdf_document",
    output = "html_document", toc = "True", keep_tex = TRUE
  ),
  multi_output = TRUE
)
```

c.teal_report *Concatenate* teal_report *objects*

Description

Concatenate teal_report objects

Usage

```
## S3 method for class 'teal_report'
c(...)
```

Arguments

...	(teal_report) objects to concatenate

Value

A teal_report object with combined teal_card elements.

code_chunk *Generate a code chunk*

Description

This function creates a code_chunk object that represents code to be displayed in a report. It stores the code of any language (see lang argument) and any specified chunk options (e.g., echo, eval).

Usage

```
code_chunk(code, ..., lang = "R")
```

Arguments

code	(character) The code to be displayed in the code chunk.
...	Additional named parameters to be included as chunk options. These control how the chunk behaves when rendered (e.g., echo = TRUE, eval = FALSE, message = FALSE). See knitr options for available options.
lang	(character(1)) The language of the code chunk. Defaults to "R". See knitr::knit_engines for supported languages (e.g., "python", "bash").

Details

Important Notes:

- The code is **not evaluated**; it is only stored as text with formatting attributes.
- When converted to output, code_chunk produces markdown code block syntax (```` ```{lang} ... ``` ````) or HTML <pre><code>...</code></pre> blocks.
- The document is **not rendered** using rmarkdown::render. The code_chunk is part of the teal_card API for building reproducible documents that are produced as-is.

Typical Workflow:

1. Create a code_chunk object with your code and options
2. Add it to a teal_card using teal_card() or c()
3. The card produces the formatted code block in the final document output

Value

An object of class code_chunk

See Also

- teal_card() for creating report cards that can contain code_chunk objects

Examples

```
# Basic code chunk with options
code_chunk("x <- 1:10", echo = TRUE, message = FALSE)

# Python code chunk
code_chunk("import pandas as pd", lang = "python", eval = FALSE)

# Code chunk with multiple knitr options
code_chunk(
  "plot(mtcars$mpg, mtcars$hp)",
  echo = TRUE,
  eval = TRUE,
  fig.width = 7,
  fig.height = 5,
  warning = FALSE
)
```

download_report_button *Download report button module*

Description

Provides a button that triggers downloading a report.

For more information, refer to the vignette: `vignette("simpleReporter", "teal.reporter")`.

Usage

```
download_report_button_ui(id, label = NULL)

download_report_button_srv(
  id,
  reporter,
  global_knitr = getOption("teal.reporter.global_knitr"),
  rmd_output = getOption("teal.reporter.rmd_output"),
  rmd_yaml_args = getOption("teal.reporter.rmd_yaml_args")
)
```

Arguments

`id`	(`character(1)`) this `shiny` module's id.
`label`	(`character(1)`) label of the button. By default it is empty.
`reporter`	(`Reporter`) instance.
`global_knitr`	(`list`) of `knitr` parameters (passed to `knitr::opts_chunk$set`) for customizing the rendering process.
`rmd_output`	(`character`) vector with `rmarkdown` output types, by default all possible `pdf_document`, `html_document`, `powerpoint_presentation`, and `word_document`. If vector is named then those names will appear in the `UI`.
`rmd_yaml_args`	(`named list`) with `Rmd yaml` header fields and their default values. This `list` will result in the custom subset of UI inputs for the download reporter functionality. Default `list(author = "NEST", title = "Report", date = Sys.Date(), output = "html_document", toc = FALSE)`. The `list` must include at least `"output"` field. The default value for `"output"` has to be in the `rmd_output` argument.

Details

To access the default values for the `global_knitr` parameter, use `getOption('teal.reporter.global_knitr')`. These defaults include:

- `echo = TRUE`
- `tidy.opts = list(width.cutoff = 60)`
- `tidy = TRUE` if `formatR` package is installed, `FALSE` otherwise

Value

NULL.

`eval_code-teal_report` *Evaluate code in* `qenv`

Description

Evaluate code in `qenv`

Usage

```
## S4 method for signature 'teal_report'
eval_code(object, code, code_block_opts = list(), ...)
```

Arguments

`object`	(`teal_report`)
`code`	(`character`, `language` or `expression`) code to evaluate. It is possible to preserve original formatting of the `code` by providing a `character` or an `expression` being a result of `parse(keep.source = TRUE)`.
`code_block_opts`	(`list`) Additional options for the R code chunk in R Markdown.
`...`	(`dots`) additional arguments passed to future methods.

Details

`eval_code()` evaluates given code in the `qenv` environment and appends it to the `code` slot. Thus, if the `qenv` had been instantiated empty, contents of the environment are always a result of the stored code.

Value

`teal_reporter` environment with the code evaluated and the outputs added to the card or `qenv.error` if evaluation fails.

See Also

within.qenv

Examples

```
td <- teal.data::teal_data()
td <- teal.code::eval_code(td, "iris <- iris")
tr <- as.teal_report(td)
tr <- teal.code::eval_code(tr, "a <- 1")
tr <- teal.code::eval_code(tr, "b <- 2L # with comment")
tr <- teal.code::eval_code(tr, quote(library(checkmate)))
tr <- teal.code::eval_code(tr, expression(assert_number(a)))
teal_card(tr)
```

---

load_report_button *Load* Reporter *button module*

Description

Provides a button to upload ReporterCard(s) to the Reporter.

For more information, refer to the vignette: `vignette("simpleReporter", "teal.reporter")`.

Usage

```
report_load_ui(id, label = NULL)

report_load_srv(id, reporter)
```

Arguments

id	character(1) this shiny module's id.
label	(character(1)) label of the button. By default it is empty.
reporter	Reporter instance.

Value

shiny::tagList

shiny::moduleServer

metadata *Access metadata from a* teal_card *or* ReportCard

Description

This function retrieves metadata from a teal_card or ReportCard object. When which is NULL, it returns all metadata fields as a list.

Usage

```
metadata(object, which = NULL)

## S3 method for class 'teal_card'
metadata(object, which = NULL)

## S3 method for class 'ReportCard'
metadata(object, which = NULL)
```

Arguments

object	(teal_card or ReportCard) The object from which to extract metadata.
which	(character or NULL) The name of the metadata field to extract.

Value

A list of metadata fields or a specific field if which is provided.

metadata<- *Set metadata for a* teal_card *or* ReportCard

Description

This function allows you to set or modify metadata fields in a teal_card or ReportCard object. It can be used to add new metadata or update existing fields.

Usage

```
metadata(object, which = NULL) <- value

## S3 replacement method for class 'teal_card'
metadata(object, which = NULL) <- value

## S3 replacement method for class 'ReportCard'
metadata(object, which) <- value
```

Arguments

object	(teal_card or ReportCard) The object to modify.
which	(character) The name of the metadata field to set.
value	The value to assign to the specified metadata field.

Details

The ReportCard class only supports the title field in metadata.

Value

The modified object with updated metadata.

---

print.rmd_yaml_header *Print method for the* yaml_header *class*

Description

Print method for the yaml_header class

Usage

```
## S3 method for class 'rmd_yaml_header'
print(x, ...)
```

Arguments

x	(rmd_yaml_header) class object.
...	optional text.

Value

NULL.

Examples

```
input <- list(author = "", output = "pdf_document", toc = TRUE, keep_tex = TRUE)
out <- as_yaml_auto(input)
out
print(out)
```

render *Render* teal_card

Description

Render teal_card

Usage

```
render(
  input,
  output_dir = getwd(),
  global_knitr = getOption("teal.reporter.global_knitr"),
  keep_rmd = TRUE,
  ...
)
```

Arguments

input	(teal_report or teal_code) object to render.
output_dir	The output directory for the rendered output_file. This allows for a choice of an alternate directory to which the output file should be written (the default output directory of that of the input file). If a path is provided with a filename in output_file the directory specified here will take precedence. Please note that any directory path provided will create any necessary directories if they do not exist.
global_knitr	(list) options to apply to every code chunk in a teal_card document. Read more here.
keep_rmd	(logical(1)) if .Rmd should be kept after rendering to desired output_format.
...	arguments passed to rmarkdown::render.

Examples

```
report <- teal_report()
teal_card(report) <- c(
  teal_card(report),
  "## Document section",
  "Lorem ipsum dolor sit amet"
)
report <- within(report, a <- 2)
report <- within(report, plot(a))
metadata(teal_card(report)) <- list(
  title = "My Document",
  author = "NEST"
)
if (interactive()) {
 render(report, output_format = rmarkdown::pdf_document(), global_knitr = list(fig.width = 10))
}
```

ReportCard ReportCard: *An* R6 *class for building report elements*

Description

[Deprecated]

This R6 class that supports creating a report card containing text, plot, table and metadata blocks that can be appended and rendered to form a report output from a shiny app.

Lifecycle

This class is deprecated. Use teal_report class instead for new implementations. See vignette("teal-report-class", "teal.reporter") for more information.

Methods

Public methods:

- ReportCard$new()
- ReportCard$append_table()
- ReportCard$append_html()
- ReportCard$append_plot()
- ReportCard$append_text()
- ReportCard$append_rcode()
- ReportCard$append_content()
- ReportCard$get_content()
- ReportCard$reset()
- ReportCard$get_metadata()
- ReportCard$append_metadata()
- ReportCard$get_name()
- ReportCard$set_name()
- ReportCard$set_content_names()
- ReportCard$to_list()
- ReportCard$from_list()
- ReportCard$clone()

Method new(): Initialize a ReportCard object.

Usage:

```
ReportCard$new()
```

Returns: Object of class ReportCard, invisibly.

Examples:

```
card <- ReportCard$new()
```

Method `append_table()`**:** Appends a table to this `ReportCard`.

Usage:

```
ReportCard$append_table(table)
```

Arguments:

`table` A (`data.frame` or `rtables` or `TableTree` or `ElementaryTable` or `listing_df`) that can be coerced into a table.

Returns: `self`, invisibly.

Examples:

```
card <- ReportCard$new()$append_table(iris)
```

Method `append_html()`**:** Appends a html content to this `ReportCard`.

Usage:

```
ReportCard$append_html(content)
```

Arguments:

`content` An object that can be rendered as a HTML content.

Returns: `self`, invisibly.

Examples:

```
card <- ReportCard$new()$append_html(shiny::div("HTML Content"))
```

Method `append_plot()`**:** Appends a plot to this `ReportCard`.

Usage:

```
ReportCard$append_plot(plot, dim = NULL)
```

Arguments:

`plot` (`ggplot` or `grob` or `trellis`) plot object.

`dim` (`numeric(2)`) width and height in pixels.

Returns: `self`, invisibly.

Method `append_text()`**:** Appends a text paragraph to this `ReportCard`.

Usage:

```
ReportCard$append_text(
  text,
  style = c("default", "header2", "header3", "verbatim")
)
```

Arguments:

`text` (`character`) The text content to add.

`style` (`character(1)`) the style of the paragraph.

Returns: `self`, invisibly.

Examples:

```
card <- ReportCard$new()$append_text("A paragraph of default text")
```

Method `append_rcode()`: Appends an R code chunk to `ReportCard`.

Usage:

```
ReportCard$append_rcode(text, ...)
```

Arguments:

`text` (`character`) The R code to include.

`...` Additional `rmarkdown` parameters for formatting the R code chunk.

Returns: `self`, invisibly.

Examples:

```
card <- ReportCard$new()$append_rcode("2+2", echo = FALSE)
```

Method `append_content()`: Appends a generic content to this `ReportCard`.

Usage:

```
ReportCard$append_content(content)
```

Arguments:

`content` (Object.)

Returns: `self`, invisibly.

Examples:

```
card <- ReportCard$new()$append_content(code_chunk("foo <- 2"))
```

Method `get_content()`: Get all content blocks from this `ReportCard`.

Usage:

```
ReportCard$get_content()
```

Returns: `teal_card()` containing appended elements.

Examples:

```
card <- ReportCard$new()$append_text("Some text")$append_metadata("rc", "a <- 2 + 2")

card$get_content()
```

Method `reset()`: Clears all content and metadata from `ReportCard`.

Usage:

```
ReportCard$reset()
```

Returns: `self`, invisibly.

Method `get_metadata()`: Get the metadata associated with `ReportCard`.

Usage:

```
ReportCard$get_metadata()
```

Returns: `named list` list of elements.

Examples:

```
card <- ReportCard$new()$append_text("Some text")$append_metadata("rc", "a <- 2 + 2")

card$get_metadata()
```

Method `append_metadata()`**:** Appends metadata to this `ReportCard`.

Usage:

```
ReportCard$append_metadata(key, value)
```

Arguments:

`key` (`character(1)`) string specifying the metadata key.
`value` value associated with the metadata key.

Returns: `self`, invisibly.

Method `get_name()`**:** Get the name of the `ReportCard`.

Usage:

```
ReportCard$get_name()
```

Returns: `character` a card name.

Examples:

```
ReportCard$new()$set_name("NAME")$get_name()
```

Method `set_name()`**:** Set the name of the `ReportCard`.

Usage:

```
ReportCard$set_name(name)
```

Arguments:

`name` (`character(1)`) a card name.

Returns: `self`, invisibly.

Examples:

```
ReportCard$new()$set_name("NAME")$get_name()
```

Method `set_content_names()`**:** Set content block names for compatibility with newer `teal_card`

Usage:

```
ReportCard$set_content_names(new_names)
```

Arguments:

`new_names` (`character`) vector of new names.

Method `to_list()`**:** Convert the `ReportCard` to a list, including content and metadata.

Usage:

```
ReportCard$to_list(output_dir = lifecycle::deprecated())
```

Arguments:

`output_dir` (`character`) with a path to the directory where files will be copied.

Returns: (`named list`) a `ReportCard` representation.

Method `from_list()`**:** Reconstructs the `ReportCard` from a list representation.

Usage:

```
ReportCard$from_list(card, output_dir = lifecycle::deprecated())
```

Arguments:

`card` (`named list`) a `ReportCard` representation.

`output_dir` (`character`) with a path to the directory where a file will be copied.

Returns: `self`, invisibly.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(ggplot2)

card <- ReportCard$new()$append_plot(
  ggplot(iris, aes(x = Petal.Length)) + geom_histogram()
)


library(ggplot2)

card <- ReportCard$new()$append_text("Some text")$append_plot(
  ggplot(iris, aes(x = Petal.Length)) + geom_histogram()
)$append_text("Some text")$append_metadata(key = "lm",
                  value = lm(Ozone ~ Solar.R, airquality))
card$get_content()
card$get_metadata()


library(ggplot2)

card <- ReportCard$new()$append_text("Some text")$append_plot(
  ggplot(iris, aes(x = Petal.Length)) + geom_histogram()
)$append_text("Some text")$append_metadata(key = "lm",
                  value = lm(Ozone ~ Solar.R, airquality))
card$get_content()

card$to_list(tempdir())


library(ggplot2)

card <- ReportCard$new()$append_text("Some text")$append_plot(
  ggplot(iris, aes(x = Petal.Length)) + geom_histogram()
)$append_text("Some text")$append_metadata(key = "lm",
```

```
                  value = lm(Ozone ~ Solar.R, airquality))
card$get_content()

ReportCard$new()$from_list(card$to_list(tempdir()), tempdir())


## ------------------------------------------------
## Method `ReportCard$new`
## ------------------------------------------------

card <- ReportCard$new()


## ------------------------------------------------
## Method `ReportCard$append_table`
## ------------------------------------------------

card <- ReportCard$new()$append_table(iris)


## ------------------------------------------------
## Method `ReportCard$append_html`
## ------------------------------------------------

card <- ReportCard$new()$append_html(shiny::div("HTML Content"))


## ------------------------------------------------
## Method `ReportCard$append_text`
## ------------------------------------------------

card <- ReportCard$new()$append_text("A paragraph of default text")


## ------------------------------------------------
## Method `ReportCard$append_rcode`
## ------------------------------------------------

card <- ReportCard$new()$append_rcode("2+2", echo = FALSE)


## ------------------------------------------------
## Method `ReportCard$append_content`
## ------------------------------------------------

card <- ReportCard$new()$append_content(code_chunk("foo <- 2"))


## ------------------------------------------------
## Method `ReportCard$get_content`
## ------------------------------------------------

card <- ReportCard$new()$append_text("Some text")$append_metadata("rc", "a <- 2 + 2")
```

```
card$get_content()



## ------------------------------------------------
## Method `ReportCard$get_metadata`
## ------------------------------------------------

card <- ReportCard$new()$append_text("Some text")$append_metadata("rc", "a <- 2 + 2")

card$get_metadata()


## ------------------------------------------------
## Method `ReportCard$get_name`
## ------------------------------------------------

ReportCard$new()$set_name("NAME")$get_name()

## ------------------------------------------------
## Method `ReportCard$set_name`
## ------------------------------------------------

ReportCard$new()$set_name("NAME")$get_name()
```

Reporter *Reporter: An R6 class for managing reports*

Description

This R6 class is designed to store and manage reports, facilitating the creation, manipulation, and serialization of report-related data. It supports both ReportCard and teal_card objects, allowing flexibility in the types of reports that can be stored and managed.

Methods

Public methods:

- Reporter$new()
- Reporter$append_cards()
- Reporter$reorder_cards()
- Reporter$replace_card()
- Reporter$get_cards()
- Reporter$get_blocks()
- Reporter$reset()
- Reporter$remove_cards()
- Reporter$get_metadata()

- Reporter$append_metadata()
- Reporter$from_reporter()
- Reporter$to_list()
- Reporter$write_figures()
- Reporter$from_list()
- Reporter$to_jsondir()
- Reporter$from_jsondir()
- Reporter$set_id()
- Reporter$open_previewer()
- Reporter$get_cached_html()
- Reporter$get_id()
- Reporter$set_template()
- Reporter$get_template()
- Reporter$clone()

Method `new()`**:** Initialize a `Reporter` object.

Usage:

```
Reporter$new()
```

Returns: Object of class `Reporter`, invisibly.

Examples:

```
reporter <- Reporter$new()
```

Method `append_cards()`**:** Append one or more `ReportCard` or `teal_card` objects to the `Reporter`.

Usage:

```
Reporter$append_cards(cards)
```

Arguments:

`cards` (`ReportCard` or `teal_card`) or a list of such objects

Returns: `self`, invisibly.

Method `reorder_cards()`**:** Reorders `teal_card` objects in `Reporter`.
Reorders `teal_card` objects in `Reporter`.

Usage:

```
Reporter$reorder_cards(new_order)
```

Arguments:

`new_order` `character` vector with names of `teal_card` objects to be set in this order.

Returns: `self`, invisibly.

Method `replace_card()`**:** Sets `ReportCard` or `teal_card` content.

Usage:

```
Reporter$replace_card(card, card_id)
```

Arguments:

`card` The new object (`ReportCard` or `teal_card`) to replace the existing one.

`card_id` (`character(1)`) the unique id of the card to be replaced.

Returns: `self`, invisibly.

Method `get_cards()`**:** Retrieves all `teal_card` objects contained in `Reporter`.

Usage:

```
Reporter$get_cards()
```

Returns: A (`list`) of `teal_card` objects.

Method `get_blocks()`**:** Compiles and returns all content blocks from the `teal_card` objects in the `Reporter`.

Usage:

```
Reporter$get_blocks(sep = "\\newpage")
```

Arguments:

`sep` An optional separator to insert between each content block. Default is a `\n\\newpage\n` markdown.

Returns: `list()` of `teal_card`

Method `reset()`**:** Resets the `Reporter`, removing all `teal_card` objects and metadata.

Usage:

```
Reporter$reset()
```

Returns: `self`, invisibly.

Method `remove_cards()`**:** Removes specific `teal_card` objects from the `Reporter` by their indices.

Usage:

```
Reporter$remove_cards(ids = NULL)
```

Arguments:

`ids` (`integer`, `character`) the indexes of cards (either name)

Returns: `self`, invisibly.

Method `get_metadata()`**:** Get the metadata associated with this `Reporter`.

Usage:

```
Reporter$get_metadata()
```

Returns: `named list` of metadata to be appended.

Examples:

```
reporter <- Reporter$new()$append_metadata(list(sth = "sth"))
reporter$get_metadata()
```

Method `append_metadata()`**:** Appends metadata to this `Reporter`.

Usage:

```
Reporter$append_metadata(meta)
```

Arguments:

`meta` (named `list`) of metadata to be appended.

Returns: `self`, invisibly.

Examples:

```
reporter <- Reporter$new()$append_metadata(list(sth = "sth"))
reporter$get_metadata()
```

Method `from_reporter()`**:** Reinitializes a `Reporter` instance by copying the report cards and metadata from another `Reporter`.

Usage:

```
Reporter$from_reporter(reporter)
```

Arguments:

`reporter` (`Reporter`) instance to copy from.

Returns: invisibly self

Examples:

```
reporter <- Reporter$new()
reporter$from_reporter(reporter)
```

Method `to_list()`**:** Convert a `Reporter` to a list and transfer any associated files to specified directory.

Usage:

```
Reporter$to_list(output_dir)
```

Arguments:

`output_dir` (`character(1)`) a path to the directory where files will be copied.

Returns: `named list` representing the `Reporter` instance, including version information, metadata, and report cards.

Examples:

```
reporter <- Reporter$new()
tmp_dir <- file.path(tempdir(), "testdir")
dir.create(tmp_dir)
reporter$to_list(tmp_dir)
```

Method `write_figures()`**:** Extracts and saves all figure elements from the `teal_card` objects in the `Reporter` to a specified directory.

Usage:

```
Reporter$write_figures(output_dir, sub_directory = "figures")
```

Arguments:

`output_dir` (`character(1)`) a path to the directory where figures will be saved.

`sub_directory` (`character(1)`) a sub-directory within `output_dir` to save figures.

Method `from_list()`**:** Reinitializes a `Reporter` from a list representation and associated files in a specified directory.

Usage:

```
Reporter$from_list(rlist, output_dir)
```

Arguments:

`rlist` (`named list`) representing a `Reporter` instance.

`output_dir` (`character(1)`) a path to the directory from which files will be copied.

Returns: `self`, invisibly.

Examples:

```
reporter <- Reporter$new()
tmp_dir <- file.path(tempdir(), "testdir")
unlink(tmp_dir, recursive = TRUE)
dir.create(tmp_dir)
reporter$from_list(reporter$to_list(tmp_dir), tmp_dir)
```

Method `to_jsondir()`**:** Serializes the `Reporter` to a `JSON` file and copies any associated files to a specified directory.

Usage:

```
Reporter$to_jsondir(output_dir)
```

Arguments:

`output_dir` (`character(1)`) a path to the directory where files will be copied, `JSON` and statics.

Returns: `output_dir` argument.

Examples:

```
reporter <- Reporter$new()
tmp_dir <- file.path(tempdir(), "jsondir")
dir.create(tmp_dir)
reporter$to_jsondir(tmp_dir)
```

Method `from_jsondir()`**:** Reinitializes a `Reporter` from a `JSON` file and files in a specified directory.

Usage:

```
Reporter$from_jsondir(output_dir)
```

Arguments:

`output_dir` (`character(1)`) a path to the directory with files, `JSON` and statics.

Returns: `self`, invisibly.

Examples:

```
reporter <- Reporter$new()
tmp_dir <- file.path(tempdir(), "jsondir")
dir.create(tmp_dir)
unlink(list.files(tmp_dir, recursive = TRUE))
reporter$to_jsondir(tmp_dir)
reporter$from_jsondir(tmp_dir)
```

Method `set_id()`**:** Set the `Reporter` id Optionally add id to a `Reporter` which will be compared when it is rebuilt from a list. The id is added to the downloaded file name.

Usage:

```
Reporter$set_id(id)
```

Arguments:

`id` (`character(1)`) a Report id.

Returns: `self`, invisibly.

Method `open_previewer()`**:** Get or set the reactive trigger to open the previewer modal.

Usage:

```
Reporter$open_previewer(val)
```

Arguments:

`val` value to the passed to the reactive trigger.

Returns: `reactiveVal` value

Method `get_cached_html()`**:** Get cached HTML for a specific `teal_card` by its id.

Usage:

```
Reporter$get_cached_html(card_id)
```

Arguments:

`card_id` (`character(1)`) the unique id of the card.

Method `get_id()`**:** Get the `Reporter` id

Usage:

```
Reporter$get_id()
```

Returns: `character(1)` the `Reporter` id.

Method `set_template()`**:** Set template function for `teal_card` Set a function that is called on every report content (of class `teal_card`) added through `$append_cards`

Usage:

```
Reporter$set_template(template)
```

Arguments:

`template` (`function`) a template function.

Returns: `self`, invisibly.

Examples:

```
reporter <- teal.reporter::Reporter$new()
template_fun <- function(document) {
  disclaimer <- teal.reporter::teal_card("Here comes disclaimer text")
  c(disclaimer, document)
}
reporter$set_template(template_fun)
doc1 <- teal.reporter::teal_card("## Header 2 text", "Regular text")
metadata(doc1, "title") <- "Welcome card"
reporter$append_cards(doc1)
reporter$get_cards()
```

Method `get_template()`**:** Get the `Reporter` template

Usage:

```
Reporter$get_template()
```

Returns: a template `function`.

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Note

if Report has an id when converting to JSON then It will be compared to the currently available one.

if Report has an id when converting to JSON then It will be compared to the currently available one.

Examples

```
library(ggplot2)

card1 <- teal_card("## Header 2 text", "A paragraph of default text")
card1 <- c(card1, ggplot(iris, aes(x = Petal.Length)) + geom_histogram())
metadata(card1, "title") <- "Card1"

card2 <- teal_card("Document introduction")
metadata(card2, "title") <- "Card2"

reporter <- Reporter$new()
reporter$append_cards(list(card1, card2))


library(rtables)
# With the card1 from above
lyt <- analyze(split_rows_by(basic_table(), "Day"), "Ozone", afun = mean)
table_res2 <- build_table(lyt, airquality)
card2 <- teal_card(
  "## Header 2 text",
  "A paragraph of default text",
  table_res2
)
metadata(card2, "title") <- "Card2"

reporter <- Reporter$new()
reporter$append_cards(list(card1, card2))

names(reporter$get_cards())
reporter$reorder_cards(c("Card2", "Card1"))
names(reporter$get_cards())
```

```
# With card1 and card2 from above

metadata(reporter$get_cards()[[1]], "title")
reporter$replace_card(card2, names(reporter$get_cards())[[1]])
metadata(reporter$get_cards()[[1]], "title")


# With card1 and card2 from above

reporter <- Reporter$new()
reporter$append_cards(list(card1, card2))
reporter$get_cards()


# With card1 and card2 from above

reporter <- Reporter$new()
reporter$append_cards(list(card1, card2))
reporter$get_blocks()


## ------------------------------------------------
## Method `Reporter$new`
## ------------------------------------------------

reporter <- Reporter$new()


## ------------------------------------------------
## Method `Reporter$get_metadata`
## ------------------------------------------------

reporter <- Reporter$new()$append_metadata(list(sth = "sth"))
reporter$get_metadata()


## ------------------------------------------------
## Method `Reporter$append_metadata`
## ------------------------------------------------

reporter <- Reporter$new()$append_metadata(list(sth = "sth"))
reporter$get_metadata()


## ------------------------------------------------
## Method `Reporter$from_reporter`
## ------------------------------------------------

reporter <- Reporter$new()
reporter$from_reporter(reporter)

## ------------------------------------------------
## Method `Reporter$to_list`
```

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

reporter <- Reporter$new()
tmp_dir <- file.path(tempdir(), "testdir")
dir.create(tmp_dir)
reporter$to_list(tmp_dir)

## ------------------------------------------------
## Method `Reporter$from_list`
## ------------------------------------------------

reporter <- Reporter$new()
tmp_dir <- file.path(tempdir(), "testdir")
unlink(tmp_dir, recursive = TRUE)
dir.create(tmp_dir)
reporter$from_list(reporter$to_list(tmp_dir), tmp_dir)

## ------------------------------------------------
## Method `Reporter$to_jsondir`
## ------------------------------------------------

reporter <- Reporter$new()
tmp_dir <- file.path(tempdir(), "jsondir")
dir.create(tmp_dir)
reporter$to_jsondir(tmp_dir)

## ------------------------------------------------
## Method `Reporter$from_jsondir`
## ------------------------------------------------

reporter <- Reporter$new()
tmp_dir <- file.path(tempdir(), "jsondir")
dir.create(tmp_dir)
unlink(list.files(tmp_dir, recursive = TRUE))
reporter$to_jsondir(tmp_dir)
reporter$from_jsondir(tmp_dir)

## ------------------------------------------------
## Method `Reporter$set_template`
## ------------------------------------------------


reporter <- teal.reporter::Reporter$new()
template_fun <- function(document) {
  disclaimer <- teal.reporter::teal_card("Here comes disclaimer text")
  c(disclaimer, document)
}
reporter$set_template(template_fun)
doc1 <- teal.reporter::teal_card("## Header 2 text", "Regular text")
metadata(doc1, "title") <- "Welcome card"
reporter$append_cards(doc1)
reporter$get_cards()
```

reporter_previewer *Show report previewer button module*

Description

[Experimental] Provides a button that triggers showing the report preview in a modal.

For more details see the vignette: `vignette("previewerReporter", "teal.reporter")`.

Usage

```
preview_report_button_ui(id, label = "Preview Report")

preview_report_button_srv(id, reporter)
```

Arguments

`id`	(`character(1)`) `shiny` module instance id.
`label`	(`character(1)`) label of the button. By default it is "Preview Report".
`reporter`	(`Reporter`) instance.

Value

`NULL`.

reporter_previewer_deprecated *Report previewer module*

Description

[Deprecated]

Module offers functionalities to visualize, manipulate, and interact with report cards that have been added to a report. It includes a previewer interface to see the cards and options to modify the report before downloading.

Cards are saved by the `shiny` bookmarking mechanism.

For more details see the vignette: `vignette("previewerReporter", "teal.reporter")`.

This function is deprecated and will be removed in the next release. Please use `preview_report_button_ui()` and `preview_report_button_srv()` to create a preview button that opens a modal with the report preview.

Usage

```
reporter_previewer_ui(id)

reporter_previewer_srv(
  id,
  reporter,
  global_knitr = getOption("teal.reporter.global_knitr"),
  rmd_output = getOption("teal.reporter.rmd_output"),
  rmd_yaml_args = getOption("teal.reporter.rmd_yaml_args"),
  previewer_buttons = c("download", "load", "reset")
)
```

Arguments

id	(character(1)) shiny module instance id.
reporter	(Reporter) instance.
global_knitr	(list) of knitr parameters (passed to knitr::opts_chunk$set) for customizing the rendering process.
rmd_output	(character) vector with rmarkdown output types, by default all possible pdf_document, html_document, powerpoint_presentation, and word_document. If vector is named then those names will appear in the UI.
rmd_yaml_args	(named list) with Rmd yaml header fields and their default values. This list will result in the custom subset of UI inputs for the download reporter functionality. Default list(author = "NEST", title = "Report", date = Sys.Date(), output = "html_document", toc = FALSE). The list must include at least "output" field. The default value for "output" has to be in the rmd_output argument.
previewer_buttons	(character) set of modules to include with c("download", "load", "reset") possible values and "download" is required. Default c("download", "load", "reset")

Details

To access the default values for the global_knitr parameter, use getOption('teal.reporter.global_knitr'). These defaults include:

- echo = TRUE
- tidy.opts = list(width.cutoff = 60)
- tidy = TRUE if formatR package is installed, FALSE otherwise

Value

NULL.

reset_report_button *Reset report button module*

Description

Provides a button that triggers resetting the report content.

For more information, refer to the vignette: `vignette("simpleReporter", "teal.reporter")`.

Usage

```
reset_report_button_ui(id, label = NULL)

reset_report_button_srv(id, reporter)
```

Arguments

id	(`character(1)`) shiny module instance id.
label	(`character(1)`) label of the button. By default `NULL`.
reporter	(`Reporter`) instance.

Value

`NULL`.

rmd_outputs *Get document output types from the* rmarkdown *package*

Description

Retrieves vector of available document output types from the `rmarkdown` package, such as `pdf_document`, `html_document`, etc.

Usage

```
rmd_outputs()
```

Value

`character` vector.

Examples

```
rmd_outputs()
```

rmd_output_arguments *Get document output arguments from the* rmarkdown *package*

Description

Retrieves the arguments for a specified document output type from the rmarkdown package.

Usage

```
rmd_output_arguments(output_name, default_values = FALSE)
```

Arguments

output_name	(character) rmarkdown output name.
default_values	(logical(1)) if to return a default values for each argument.

Examples

```
rmd_output_arguments("pdf_document")
rmd_output_arguments("pdf_document", TRUE)
```

simple_reporter *Simple reporter module*

Description

Module provides compact UI and server functions for managing a report in a shiny app. This module combines functionalities for adding cards to a report, downloading the report, and resetting report content.

For more details see the vignette: vignette("simpleReporter", "teal.reporter").

Usage

```
simple_reporter_ui(id)

simple_reporter_srv(
  id,
  reporter,
  card_fun,
  global_knitr = getOption("teal.reporter.global_knitr"),
  rmd_output = getOption("teal.reporter.rmd_output"),
  rmd_yaml_args = getOption("teal.reporter.rmd_yaml_args")
)
```

Arguments

id	(character(1)) shiny module instance id.
reporter	(Reporter) instance.
card_fun	(reactive or function) which returns a teal_card or ReportCard instance.
global_knitr	(list) a global knitr parameters for customizing the rendering process.
rmd_output	(character) vector with rmarkdown output types, by default all possible pdf_document, html_document, powerpoint_presentation, and word_document. If vector is named then those names will appear in the UI.
rmd_yaml_args	(named list) with Rmd yaml header fields and their default values. This list will result in the custom subset of UI inputs for the download reporter functionality. Default list(author = "NEST", title = "Report", date = Sys.Date(), output = "html_document", toc = FALSE). The list must include at least "output" field. The default value for "output" has to be in the rmd_output argument.

Details

To access the default values for the global_knitr parameter, use getOption('teal.reporter.global_knitr'). These defaults include:

- echo = TRUE
- tidy.opts = list(width.cutoff = 60)
- tidy = TRUE if formatR package is installed, FALSE otherwise

Value

NULL.

Examples

```
if (interactive()) {
  library(shiny)

  shinyApp(
    ui = fluidPage(simple_reporter_ui("simple")),
    server = function(input, output, session) {
      simple_reporter_srv("simple", Reporter$new(), function(card) card)
    }
  )
}
```

teal_card *teal_card: An S3 class for managing teal reports*

Description

[Experimental]

The teal_card S3 class provides functionality to store, manage, edit, and adjust report contents. It enables users to create, manipulate, and serialize report-related data efficiently.

The teal_card() function serves two purposes:

1. When called with a teal_report object, it acts as a getter and returns the card slot.
2. When called with other arguments, it creates a new teal_card object from those arguments.

This function ensures that input is converted to a teal_card object. It accepts various input types and converts them appropriately.

Usage

```
teal_card(...)

teal_card(x) <- value

as.teal_card(x)

## S3 method for class 'teal_card'
c(...)

## S3 method for class 'teal_card'
x[i]
```

Arguments

...	Elements from which teal_card will be combined.
x	Object to convert to teal_card
value	(teal_card) object to set in the teal_report.
i	index specifying elements to extract or replace

Details

The teal_card class supports c() and x[i] methods for combining and subsetting elements. However, these methods only function correctly when the first element is a teal_card.

Value

An S3 list of class teal_card.

A teal_card object

See Also

code_chunk(), render(), toHTML()

Examples

```
# create an empty card
report <- teal_card()

# Create a card with content
report <- teal_card(
  "## Headline",
  "This is `iris` table",
  code_chunk("print(iris)", lang = "R"),
  iris
)

# Add elements to the report
report <- c(
  report,
  list("## mtcars Table"),
  code_chunk("print(mtcars)", lang = "R"),
  mtcars
)

# Subset the report to keep only the first two elements
report[1:2]

# Replace element
report[[1]] <- "## Iris Table"

# Append element
report <- append(report, teal_card("# Awesome Report"), after = 0)
tools::toHTML(report)

if (interactive()) {
  render(report, output_format = rmarkdown::pdf_document())
}
```

---

teal_report *Comprehensive data integration function for* teal *applications*

Description

[Stable]

Initializes a reportable data for teal application.

Usage

```
teal_report(
  ...,
  teal_card = NULL,
  code = character(0),
  join_keys = teal.data::join_keys()
)

as.teal_report(x)
```

Arguments

...	any number of objects (presumably data objects) provided as name = value pairs.
teal_card	(teal_card) object containing the report content.
code	(character, language) optional code to reproduce the datasets provided in .... Note this code is not executed and the teal_data may not be reproducible Use verify() to verify code reproducibility.
join_keys	(join_keys or single join_key_set) optional object with datasets column names used for joining. If empty then no joins between pairs of objects.
x	(qenv or teal_data) object to convert to teal_report.

Value

A teal_report object.

See Also

teal.data::teal_data

Examples

```
# Initialize teal_report with existing h2 header
report <- teal_report(teal_card = teal_card("## Analysis Report"))

# Use within() to execute code and add code-chunk
report <- within(report, {
  data <- iris
  summary_stats <- summary(data)
})

# Access objects created within the report
report$data
report$summary_stats

# within() automatically captures code and outputs
report <- within(report, {
  head(iris)
})
```

```
# Add arbitrary markdown content to the card
teal_card(report) <- c(
  teal_card(report),
  teal_card("### Conclusion", "The analysis is complete.")
)

# View the generated card with code chunks
teal_card(report)

# View report in HTML format
tools::toHTML(report)

if (interactive()) {
  # Render the report to various formats
  render(report, output_format = rmarkdown::html_document())
  render(report, output_format = rmarkdown::pdf_document())
}
```

`toHTML.default` *Convert report objects to HTML*

Description

[Experimental]

The `toHTML` S3 generic method converts various report objects into HTML representations. This is the primary method for rendering report content for display in web browsers, IDE Viewer, or for inclusion in Shiny applications.

Usage

```
## Default S3 method:
toHTML(x, ...)
```

Arguments

`x`	The object to convert to HTML. Supported types include: • `teal_card`: A list-like structure containing report elements • `teal_report`: A report object containing a `teal_card` • `ReportCard`: Deprecated R6 class for report cards • `code_chunk`: Code blocks created with `code_chunk()` • `chunk_output`: Output from evaluated code chunks • Plot objects: `ggplot`, `recordedplot`, `trellis`, `grob` • Table objects: `data.frame`, `rtables`, `TableTree`, `ElementaryTable`, `listing_df`, `gtsummary`, `flextable`, `datatables` • Text: `character` strings (rendered as markdown) • Other objects: Conditions, model summaries, etc.
`...`	Additional arguments passed to methods.

Details

Relationship with `teal_card`:

The `teal_card` class is a central component in the `teal.reporter` ecosystem. It is an S3 list where each element represents a piece of report content (text, plots, tables, code chunks, etc.). The `toHTML` method for `teal_card` objects:

1. Iterates through each element in the `teal_card` list
2. Calls `toHTML()` recursively on each element based on its class
3. Wraps all converted elements in a `bslib::card()` container

This hierarchical conversion allows complex report structures to be rendered as styled HTML with proper formatting for each content type.

Content Type Conversions:

Text and Markdown: Character strings are converted to HTML using `CommonMark` markdown syntax. Supports headers, lists, code blocks, emphasis, and other markdown features.

Code Chunks: Created with `code_chunk()`, these are rendered as collapsible Bootstrap accordions with syntax highlighting. The accordion includes the programming language indicator and an icon.

Plots: Plot objects (`ggplot`, `recordedplot`, `trellis`, `grob`) are converted to PNG images with base64-encoded data URIs, making them self-contained in the HTML output.

Tables: Table objects are converted to styled HTML tables, typically via `flextable` for consistent formatting.

Viewer Integration:

All HTML output is wrapped with `htmltools::browsable()`, which enables:

- Automatic render in IDE Viewer when displayed interactively
- Proper HTML dependency injection (Bootstrap CSS/JavaScript, Font Awesome icons, etc.)
- Standalone HTML files with all required resources

You can override the `browsable` behavior with:

```
print(toHTML(x), browse = FALSE)  # Print markup to console instead
```

Value

An HTML representation of the input object. The exact return type depends on the input class:

- For `teal_card`: A `bslib::card()` containing all elements
- For `code_chunk`: A `bslib::accordion()` with the code
- For plots: A `shiny::tags$img()` tag
- For text: HTML markup from markdown conversion
- For tables: HTML table elements

All returns are wrapped with `htmltools::browsable()` to enable viewer display.

See Also

- `teal_report()` for creating report objects
- `teal_card()` for creating report cards
- `code_chunk()` for creating code blocks
- `render()` for rendering complete reports to files

Examples

```
# Initialize empty report
report <- teal_report()

# Add arbitrary markdown elements to the report's teal_card
teal_card(report) <- c(
  teal_card(report),
  "## Document section",
  "Lorem ipsum dolor sit amet"
)

# Use within() to execute code and add code-chunk
report <- within(report, a <- 2)

# within() automatically captures code and outputs
report <- within(report, plot(a))

html <- tools::toHTML(report)
# display HTML markup in viewer
html

# Print HTML markup to console instead of viewer
print(html, browse = FALSE)
```

---

to_rmd *Convert* ReporterCard/teal_card *content to* rmarkdown

---

Description

This is an S3 generic that is used to generate content in `rmarkdown` format from various types of blocks in a `ReporterCard` or `teal_card` object.

Usage

```
to_rmd(block, ...)
```

Arguments

`block`	(any) content which can be represented in Rmarkdown syntax.
`...`	additional arguments passed to implemented methods for different classes.

Details

Customize `to_rmd`**:**

The methods for this S3 generic can be extended by the app developer or even overwritten. For this a function with the name `to_rmd.<class>` should be defined in the Global Environment or registered as an S3 method, where `<class>` is the class of the object to be converted.

For example, to override the default behavior for `code_chunk` class, you can use:

```
to_rmd.code_chunk <- function(block, ...) {
  # custom implementation
  sprintf("### A custom code chunk\n\n```{r}\n%s\n```\n", block)
}
```

Alternatively, the S3 method can be registered using `registerS3method("to_rmd", "<class>", fun)`

Defaults:

`teal.reporter` provides default `to_rmd` methods for several common classes that returns the content in appropriate R Markdown syntax. These include:

- `character`
- `code_chunk()` objects
- `ggplot2` plots
- `data.frame`
- `flextable`
- `rtables` tables
- and others.

All of these defaults can be overridden by defining new `to_rmd.<class>` methods. These methods are implemented internally using the helper function `.to_rmd.<class>`.

Value

`character(1)` containing a content or Rmarkdown document.

Examples

```
to_rmd(c("## This is a simple text block.", "", "With a paragraph break."))
to_rmd(code_chunk("summary(cars)"))
to_rmd(data.frame(x = 1:10, y = 21:30), folder_path = tempdir())


# Example with ggplot2 will create a temporary RDS file in the tempdir()
to_rmd(
  ggplot2::ggplot(mtcars, ggplot2::aes(x = wt, y = mpg)) +
    ggplot2::geom_point(),
  folder_path = tempdir() # internal argument of ggplot2 method
)
```

ui_editor_block *UI and Server functions for editing report document blocks*

Description

These functions provide a user interface and server logic for editing and extending the editor functionality to support new data types.

Usage

```
ui_editor_block(id, value, ...)

srv_editor_block(id, value, ...)
```

Arguments

id	(character(1)) A unique identifier for the module.
value	The content of the block to be edited. It can be a character string or other types.
...	Additional arguments passed to dispatch functions.

Details

The methods for this S3 generic can be extended by the app developer to new classes or even overwritten. For this a function with the name srv_editor_block.<class> and/or ui_editor_block.<class> should be defined in the Global Environment, where <class> is the class of the object to be used in the method.

For example, to override the default behavior for character class, you can use:

```
ui_editor_block.character <- function(id, value) {
  # custom implementation
  shiny::tagList(
   shiny::tags$h6(shiny::icon("pencil", class = "text-muted"), "Editable CUSTOM markdown block"),
   shiny::textAreaInput(ns("content"), label = NULL, value = value, width = "100%")
  )
}
srv_editor_block.character <- function(id, value) {
 # custom implementation
 # ...
}
```

Alternatively, you can register the S3 method using registerS3method("ui_editor_block", "<class>", fun) and registerS3method("srv_editor_block", "<class>", fun).

Optional arguments of ui_editor_block**:**

- cached_html: (shiny.tag or shiny.tag.list) Cached HTML content to display in the UI that is rendered at the time the card is being added. Default is NULL.
  teal will call on ui_editor_block with the contents of each element of the card (value argument) and an optional parameter cached_html. This parameter is not part of the S3 generic as it is optional when overriding the method.
  The usage of this argument improve the UI performance by avoiding re-rendering on the fly. When overriding this method, the usage of this argument is optional, hence it is not part of the S3 generic.

Index