Package 'quartose'

May 9, 2026

Title Dynamically Generate Quarto Syntax

Version 0.1.0

Description Provides helper functions to work programmatically within a quarto document. It allows the user to create section headers, tabsets, divs, and spans, and formats these objects into quarto syntax when printed into a document.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

URL https://github.com/djnavarro/quartose, https://quartose.djnavarro.net/

BugReports https://github.com/djnavarro/quartose/issues

Imports cli, knitr, purrr, rlang, utils

Suggests ggplot2, quarto, rmarkdown, spelling, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/Needs/website rmarkdown

Language en-US

NeedsCompilation no

Author Danielle Navarro [aut, cre, cph]

Maintainer Danielle Navarro <djnavarro@protonmail.com>

Repository CRAN

Date/Publication 2025-07-09 12:50:02 UTC

Contents

quarto_format *Format a quarto object*

Description

Creates a formatted representation of a quarto object in a form suitable for printing. When calling `knitr::knit_print()` on a quarto object, the relevant `format()` method is called first, and the formatted version is printed to the document. Note that the base `print()` method for quarto objects does not call `format()`.

Usage

```
## S3 method for class 'quarto_section'
format(x, ...)

## S3 method for class 'quarto_tabset'
format(x, ...)

## S3 method for class 'quarto_div'
format(x, ...)

## S3 method for class 'quarto_span'
format(x, ...)

## S3 method for class 'quarto_markdown'
format(x, ...)

## S3 method for class 'quarto_group'
format(x, ...)
```

Arguments

x	A quarto object.
...	Other arguments (ignored).

Details

The intent behind the `format()` methods for quarto objects is to create a ready-to-print representation of that is almost identical to what will be printed into the quarto document when `knitr::knit_print()` is called. Because of this, the formatted version of a quarto object is a string or a list of strings, but it may also include plot objects that have not yet been rendered. The resulting representation isn't always very pretty, though it is generally fairly readable.

Value

A formatted quarto object. For `quarto_section`, `quarto_span`, and `quarto_markdown` objects, the formatted output is always a string (character vector of length 1). For `quarto_tabset` and

quarto_group objects, the output is always a list whose elements are either strings or plot objects. For quarto_div objects, the output is currently a string, but this may change to list output in future if divs are permitted to contain plots.

Examples

```
# formatted sections, spans and divs ---------------------------------
sec <- quarto_section("Header", level = 2L)
spn <- quarto_span("Content", class = "underline")
div <- quarto_div("Content", class = "content-margin")

format(sec)

format(spn)

format(div)

# formatted tabsets --------------------------------------------------
tbs <- quarto_tabset(
  content = list(tab1 = 1:10, tab2 = "hello"),
  title = "Header",
  level = 2L
)

format(tbs)

# formatted groups and markdown --------------------------------------

mkd <- quarto_markdown(list("- this is a", "- markdown list"), sep = "\n")
gps <- quarto_group(list(div, mkd))

format(mkd)

format(gps)
```

---

quarto_object *Dynamically generate quarto syntax*

Description

Define quarto objects for insertion into a document. Intended to be used inside a quarto document, within a knitr code chunk with the `results: asis` option set.

Usage

```
quarto_section(title, level)

quarto_tabset(content, level, title = NULL, names = NULL)
```

```
quarto_div(content, class = NULL, sep = "")

quarto_span(content, class = NULL, sep = "")

quarto_group(content, sep = "")

quarto_markdown(content, sep = "")
```

Arguments

`title`	Character string specifying the text to use as a section title. For `quarto_section()` this is a required argument. For `quarto_tabset()` it is permitted to use `title = NULL`, in which case the tabset will be printed without a section header above it. This is the default behavior for tabsets.
`level`	Numeric header level applied to section title or tabset names. The `level` argument must be a whole number between 1 and 6. Only relevant to quarto objects that produce section headings, specifically `quarto_section()` and `quarto_tabset()`.
`content`	List or character vector containing content to be included within the quarto object. The expected format of the `content` argument differs slightly depending on which function is used. See the "details" section for more information.
`names`	Character vector of names to be applied to the tabs in a tabset. Only relevant to `quarto_tabset()`. If `names = NULL`, the names will be taken from the names of the `content` argument.
`class`	Character vector specifying CSS classes to be applied to the content. Only relevant to `quarto_div()` and `quarto_span()`. Defaults to `class = NULL`, in which case the formatted text written to the document will have a dummy CSS class "quartose-null" applied.
`sep`	Character string specifying the separator to be used when merging content for printing to the document. Defaults to `sep = ""` for all functions.

Details

The purpose of these functions is to allow the user to dynamically generate quarto syntax from R. When used within a quarto document they allow the user to generate callouts, margin text, tabsets, section headers, and other kinds of quarto output. At the current state of development the functionality is somewhat limited, discussed below.

The `quarto_*()` functions supplied by the quartose package have a common design: argument values supplied by the user are stored internally as a list, with only a minimum of processing done at the time that the function is called. The object is assigned to two S3 classes, the "quarto_object" shared by all objects, and a specific class associated with the calling function. These objects can be inspected and manipulated programmatically like any other R objects prior to printing.

When creating a quarto object, note that most `quarto_*()` functions take a `content` argument, which differs slightly depending on the context:

- For `quarto_section()` there is no ‘content“ argument: section headers have titles, but they do not contain content.
- For `quarto_span()` the ‘content“ argument *must* be a character vector, not a list.

- For `quarto_div()` the `content`` argument is permitted to be a `character` vector or a `list`, but it will a
cannot handle plot objects, but functionality may be extended to permit this in future.
- For `quarto_tabset()` the `content` argument *must* be a list. The list elements can be any printable R object: each element of the list will appear in its own tab. At present the support for graphics objects is limited: ggplot2 objects are captured and will only be rendered when `knitr::knit_print()` is called. No attempt is made (as yet!) to support other kinds of graphic objects, and if these are passed via the `content` argument the function will likely fail.
- For `quarto_markdown()` the `content` argument may be a character vector or a list of character vectors. The function will throw an error if other kinds of objects are passed via `content`.
- For `quarto_group()` the `content` argument *must* be a list, and all elements of the list must be quarto objects. The intended use of this function is simply to collect several quarto objects into a single group that will be printed all at the same time rather than sequentially.

Creating a quarto object only defines the data structure, it does not perform any formatting. Similarly, if the object is printed using `print()`, no formatting will be applied. A brief summary of the data structure will be printed to the console, no more. However, when `knitr::knit_print()` is called, the quarto object is first passed to the relevant `format()` method, which is responsible for constructing the appropriate quarto syntax. Calling `format()` will return a character vector or a list. If it returns a list all elements will either be character strings with the appropriate quarto syntax, or a plot object that has not yet been rendered. After formatting is applied the `knitr::knit_print()` method will pass the strings (or plots) to the document. For more detail on the formatting and printing methods see `knit_print.quarto_object()` and `format.quarto_object()`.

Value

These functions always return an object with parent S3 class "quarto_object", in addition to a specific S3 class corresponding to the function. For example, `quarto_section()` objects also possess the "quarto_section" class.

Examples

```
# quarto_section -------------------------------------------------------

sec <- quarto_section("A level-two header", level = 2L)

# quarto objects have two classes, a general purpose class shared by
# all quarto objects, and a class specific to the function
class(sec)

# base::print() displays an abstract summary of the object
print(sec)

# knitr::knit_print() produces the rendered quarto syntax
knitr::knit_print(sec)

# quarto_span ----------------------------------------------------------

spn1 <- quarto_span("This is plain text")
spn2 <- quarto_span("This is underlined text", class = "underline")
```

```
print(spn1)

print(spn2)

knitr::knit_print(spn1)

knitr::knit_print(spn2)

# quarto_div ------------------------------------------------------------

# quarto_div objects are flexible: they can take a character vector as
# the content argument, but can also take lists of other objects; note
# that internally the content is always represented as a list
div1 <- quarto_div("This is a callout note", class = "callout-note")
div2 <- quarto_div(
  content = list(
    quarto_span(content = "You can wrap multiple spans in a div so that"),
    quarto_span(content = "some text is highlighted", class = "mark"),
    quarto_span(content = "and some is underlined", class = "underline")
  ),
  class = c("column-margin", "callout-tip"),
  sep = " "
)

print(div1)

print(div2)

knitr::knit_print(div1)

knitr::knit_print(div2)

# quarto_tabset ---------------------------------------------------------

tbs <- quarto_tabset(list(tab1 = 1:10, tab2 = "hello"), level = 3L)

print(tbs)

knitr::knit_print(tbs)

# quarto_markdown -------------------------------------------------------

mkd <- quarto_markdown(list("- a markdown", "- list"), sep = "\n")

print(mkd)

knitr::knit_print(mkd)

# quarto_group ----------------------------------------------------------

grp <- quarto_group(list(
  quarto_div("This is a callout note", class = "callout-note"),
  quarto_div("This is a callout tip", class = "callout-tip")
```

```
))

print(grp)

knitr::knit_print(grp)
```

`quarto_print` *Print a quarto object*

Description

Prints a quarto object. When calling `knitr::knit_print()` on a quarto object, the relevant `format()` method is called first, and the formatted version is printed to the document. When calling `print()`, a summary of the object structure is printed.

Usage

```
## S3 method for class 'quarto_object'
knit_print(x, ...)

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

Arguments

`x`	A quarto object.
`...`	Other arguments (ignored).

Details

There are two print methods supplied for quarto objects, one for `base::print()` and another for `knitr::knit_print()`. The regular print method behaves similarly to any other print method: it prints a summary of the object to the R console, and invisibly returns the object itself.

When `knitr::knit_print()` is called on a quarto object, the behavior is quite different. The object is first passed to `format()`, which constructs the required quarto syntax, then the object is printed to the document (or console, if called interactively) using the appropriate syntax. In this case, the function invisibly returns `NULL`.

Value

`knitr::knit_print()` invisibly returns `NULL`; `print()` invisibly returns the quarto object itself.

Examples

```
# a quarto_section object
sec <- quarto_section("A level-two header", level = 2L)

# base::print() displays a summary of the object
print(sec)

# knitr::knit_print() displays the rendered quarto syntax
knitr::knit_print(sec)

# a quarto_span object
spn <- quarto_span("This is underlined", class = "underline")

print(spn)

knitr::knit_print(spn)
```

Index