

Package ‘quartose’

September 4, 2026

Title Dynamically Generate 'Quarto' Syntax

Version 0.2.1

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 8.0.0

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

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

Imports cli, grDevices, grid, knitr, purrr, rlang, utils

Suggests dplyr, flextable, ggplot2, lattice, patchwork, pkgdown,
quarto, rmarkdown, spelling, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/Needs/website djnavarro/waeponwifestre

Language en-US

NeedsCompilation no

Author Danielle Navarro [aut, cre, cph] (ORCID:
<<https://orcid.org/0000-0001-7648-6578>>)

Maintainer Danielle Navarro <djnavarro@protonmail.com>

Repository CRAN

Date/Publication 2026-09-04 04:10:02 UTC

Contents

as_quarto_graphic	2
quarto_format	3
quarto_object	5
quarto_print	8

as_quarto_graphic	<i>Tag an object as a graphic for quarto_tabset()/quarto_div()</i>
-------------------	--

Description

`quarto_tabset()` and `quarto_div()` auto-detect several kinds of graphics objects (ggplot2/patchwork, base R recorded plots, grid grobs, and lattice/trellis objects) and render them as images rather than as captured text output. If content includes a graphics object from a package quartose doesn't know about, wrap it in `as_quarto_graphic()` to force this treatment.

Usage

```
as_quarto_graphic(x)
```

Arguments

x	An object to tag as a graphic.
---	--------------------------------

Details

Objects tagged this way are rendered via a generic fallback: a PNG device is opened and `print(x)` is called on the (untagged) object, on the assumption that printing it draws a plot to the active graphics device. This works for many graphics-producing S3 objects but is best-effort; if `print(x)` doesn't draw anything, the resulting image will be blank.

Value

x, with the additional class "quartose_graphic" prepended.

Examples

```
# a hypothetical object whose print method draws a plot, but which
# quartose doesn't otherwise recognize as a graphic
obj <- structure(list(), class = "some_custom_plot_class")
tagged <- as_quarto_graphic(obj)
class(tagged)
```

quarto_format	<i>Format a quarto object</i>
---------------	-------------------------------

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

<code>x</code>	A quarto object.
<code>...</code>	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.

Escaping policy. `quarto_tabset()` accepts arbitrary R objects as content and captures their default printed representation (via `knitr::knit_print()/capture.output()`) to display inside each tab. Most objects' `knit_print()/print()` methods write this representation as a side effect (for example, an `lm` object's `Call: summary`, or a data frame's default print), and that captured text may

incidentally contain `<` or `>` (for example, a tibble's `<fct>` column-type tag), which would otherwise be parsed as an unrecognized HTML tag by quarto/pandoc and silently dropped from the rendered document. To prevent this, `format.quarto_tabset()` escapes `<` and `>` to `<` and `>`; in that captured text before it is written out, and wraps it in `<pre>` tags. Some objects instead *return* their output marked via `knitr::asis_output()` (class `"knit_asis"`) rather than printing it — this includes `knitr::kable(format = "html")`, flextable objects, gt tables, and most htmlwidget-like objects. For these, `format.quarto_tabset()` detects the `"knit_asis"` class and emits the raw markup unescaped and without a `<pre>` wrapper, so it is rendered as intended (a table, widget, etc.) rather than displayed as literal text. `quarto_div()` supports the same `"knit_asis"` passthrough for its content, emitted unescaped for the same reason — but unlike `quarto_tabset()`, it validates content at construction time, so objects that neither print like ordinary R objects nor return `"knit_asis"` output (e.g. a bare list, a model object, or a number) are rejected up front rather than falling back to captured/escaped text. `quarto_span()` content is restricted by validation to character vectors (never arbitrary captured print output), and `quarto_markdown()` is explicitly meant to carry raw markdown/HTML through untouched — neither of these is escaped.

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 a string unless content includes a graphics object, in which case it is a list whose elements are either strings or plot objects, exactly as for `quarto_tabset`.

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 <code>quarto_section()</code> this is a required argument. For <code>quarto_tabset()</code> it is permitted to use <code>title = NULL</code> , 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 <code>level</code> argument must be a whole number between 1 and 6. Only relevant to quarto objects that produce section headings, specifically <code>quarto_section()</code> and <code>quarto_tabset()</code> .
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 <code>quarto_tabset()</code> . If <code>names = NULL</code> , 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 <code>quarto_div()</code> and <code>quarto_span()</code> . Defaults to <code>class = NULL</code> , in which case the formatted text written to the document will have a dummy CSS class "quarto-null" applied.
sep	Character string specifying the separator to be used when merging content for printing to the document. Defaults to <code>sep = ""</code> 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 `quarto` 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 always be stored internally as a list. If the input is a list, it can contain other quarto objects, graphics objects of the same kinds supported by `quarto_tabset()` (`ggplot2`/`patchwork`, base R recorded plots, grid grobs, lattice/trellis objects, or anything tagged via `as_quarto_graphic()`), or objects whose `knitr::knit_print()` output is marked via `knitr::asis_output()` (e.g. `knitr::kable(format = "html")`, `flextable`, `gt`) – these are rendered as their raw HTML rather than being coerced through `format()`. Anything else (e.g. a bare list, a model object, or a number) is rejected at construction time. The intended use for this is a div that contains several spans, but it is not limited to this use case.
- 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. Several kinds of graphics objects are auto-detected and rendered as images rather than as captured text: `ggplot2` (and `patchwork`, which subclasses `ggplot2`) objects, base R recorded plots (`grDevices::recordPlot()`), grid grobs, and lattice/trellis objects. If a graphics object from another package isn't auto-detected, wrap it with `as_quarto_graphic()` to force this treatment. Rendering happens when `knitr::knit_print()` is called.
- 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 `knitr::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

<code>x</code>	A quarto object.
<code>...</code>	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`.

Note that `print()` is console-only: it never writes quarto syntax into the rendered document, even when called from within a code chunk. Its console summary is written via `cli::cli_text()`, which routes through `message()` rather than `cat()` when not attached to an interactive terminal (as is the case during quarto render). Some output formats hide message/warning chunk output by default (e.g. revealjs presentations), so a stray `print()` call left in a chunk may appear to produce nothing at all under those formats, even though the object was evaluated correctly. To emit the actual quarto syntax into the document, use `knitr::knit_print()` inside a chunk with the `results: asis` option.

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

`as_quarto_graphic`, 2

`cli::cli_text()`, 9

`format.quarto_div` (`quarto_format`), 3

`format.quarto_group` (`quarto_format`), 3

`format.quarto_markdown` (`quarto_format`), 3

`format.quarto_object` (`quarto_format`), 3

`format.quarto_section` (`quarto_format`), 3

`format.quarto_span` (`quarto_format`), 3

`format.quarto_tabset` (`quarto_format`), 3

`knit_print.quarto_div` (`quarto_print`), 8

`knit_print.quarto_group` (`quarto_print`), 8

`knit_print.quarto_markdown` (`quarto_print`), 8

`knit_print.quarto_object` (`quarto_print`), 8

`knit_print.quarto_section` (`quarto_print`), 8

`knit_print.quarto_span` (`quarto_print`), 8

`knit_print.quarto_tabset` (`quarto_print`), 8

`print.quarto_div` (`quarto_print`), 8

`print.quarto_group` (`quarto_print`), 8

`print.quarto_markdown` (`quarto_print`), 8

`print.quarto_object` (`quarto_print`), 8

`print.quarto_section` (`quarto_print`), 8

`print.quarto_span` (`quarto_print`), 8

`print.quarto_tabset` (`quarto_print`), 8

`quarto_div` (`quarto_object`), 5

`quarto_format`, 3

`quarto_group` (`quarto_object`), 5

`quarto_markdown` (`quarto_object`), 5

`quarto_object`, 5

`quarto_print`, 8

`quarto_section` (`quarto_object`), 5

`quarto_span` (`quarto_object`), 5

`quarto_tabset` (`quarto_object`), 5