Package 'quarto'

May 9, 2026

Title R Interface to 'Quarto' Markdown Publishing System

Version 1.5.1

Description Convert R Markdown documents and 'Jupyter' notebooks to a variety of output formats using 'Quarto'.

License MIT + file LICENSE

URL https://github.com/quarto-dev/quarto-r, https://quarto-dev.github.io/quarto-r/

BugReports https://github.com/quarto-dev/quarto-r/issues

Depends R (>= 4.1.0)

Imports cli, fs, htmltools, jsonlite, later, lifecycle, processx, rlang, rmarkdown, rstudioapi, tools, utils, xfun, yaml (>= 2.3.10)

Suggests bslib, callr, curl, dplyr, flextable, ggiraph, ggplot2, gt, heatmaply, kableExtra, knitr, palmerpenguins, patchwork, pkgload, plotly, rsconnect (>= 0.8.26), testthat (>= 3.1.7), thematic, tidyverse, tinytable, whoami, withr

VignetteBuilder quarto

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

SystemRequirements Quarto command line tool (<https://github.com/quarto-dev/quarto-cli>).

NeedsCompilation no

Author JJ Allaire [aut] (ORCID: <https://orcid.org/0000-0003-0174-9868>), Christophe Dervieux [cre, aut] (ORCID: <https://orcid.org/0000-0003-4474-2498>), Posit Software, PBC [cph, fnd], Gordon Woodhull [ctb]

Maintainer Christophe Dervieux <cderv@posit.co>

Repository CRAN

Date/Publication 2025-09-04 15:30:02 UTC

Contents

`add_spin_preamble` *Add spin preamble to R script*

Description

Adds a minimal spin preamble to an R script file if one doesn't already exist. The preamble includes a title derived from the filename and is formatted as a YAML block suitable preprended with `#'` for `knitr::spin()`.

Usage

```
add_spin_preamble(script, title = NULL, preamble = NULL, quiet = FALSE)
```

Arguments

`script`	Path to the R script file
`title`	Custom title for the preamble. If provided, overrides any title in the `preamble` list. If NULL, uses `preamble$title` or filename as fallback.
`preamble`	Named list of YAML metadata to include in preamble. The `title` parameter takes precedence over `preamble$title` if both are provided.
`quiet`	If `TRUE`, suppresses messages and warnings.

Details

This is useful to prepare R scripts for use with Quarto Script rendering support. See https://quarto.org/docs/computations/render-scripts.html#knitr

Value

Invisibly returns the script path if modified, otherwise invisible NULL

Preamble format

For a script named `analysis.R`, the function adds this preamble by default:

```
#' ---
#' title: analysis
#' ---
#'

# Original script content starts here
```

This is the minimal preamble required for Quarto Script rendering, so that Engine Bindings works.

Examples

```
## Not run:
# Basic usage with default title
add_spin_preamble("analysis.R")

# Custom title
add_spin_preamble("analysis.R", title = "My Analysis")

# Custom preamble with multiple fields
add_spin_preamble("analysis.R", preamble = list(
  title = "Advanced Analysis",
  author = "John Doe",
  date = Sys.Date(),
  format = "html"
))

# Title parameter overrides preamble title
add_spin_preamble("analysis.R",
```

```
  title = "Final Title",  # This takes precedence
  preamble = list(
    title = "Ignored Title",
    author = "John Doe"
  )
)

## End(Not run)
```

`check_newer_version` *Check for newer version of Quarto*

Description

Checks if a newer version of Quarto is available and informs the user about their current version status. The function compares the current Quarto version against the latest stable and prerelease versions available online.

Usage

```
check_newer_version(version = quarto_version(), verbose = TRUE)
```

Arguments

`version`	Character string specifying the Quarto version to check. Defaults to the currently installed version detected by `quarto_version()`. Use "99.9.9" to indicate a development version.
`verbose`	Logical indicating whether to print informational messages. Defaults to `TRUE`. When `FALSE`, the function runs silently and only returns the logical result.

Details

The function handles three scenarios:

- **Development version** (99.9.9): Skips version check with informational message
- **Prerelease version**: Compares against latest prerelease and informs about updates
- **Stable version**: Compares against latest stable version and suggests updates if needed

Version information is fetched from Quarto's download JSON endpoints and cached in current session for up to 24 hours to avoid repeated network requests.

Value

Invisibly returns a logical value:

- `TRUE` if an update is available
- `FALSE` if no update is needed or when using development version The function is primarily called for its side effects of printing informational messages (when `verbose = TRUE`).

Network Requirements

This function requires an internet connection to fetch the latest version information from quarto.org. If the network request fails, an error will be thrown.

See Also

`quarto_version()` for getting the current Quarto version,

Examples

```
# Check current Quarto version
check_newer_version()

# Check a specific version
check_newer_version("1.7.30")

# Check development version (will skip check)
check_newer_version("99.9.9")

# Check silently without messages
result <- check_newer_version(verbose = FALSE)
if (result) {
  message("Update available!")
}
```

---

detect_bookdown_crossrefs

Detect Bookdown Cross-References for Quarto Migration

Description

Scans R Markdown or Quarto files to identify bookdown cross-references that need to be converted to Quarto syntax. Provides detailed reports and guidance for migrating from bookdown to Quarto.

Usage

```
detect_bookdown_crossrefs(path = ".", verbose = FALSE)
```

Arguments

`path`	Character string. Path to a single `.Rmd` or `.qmd` file, or a directory containing such files. Defaults to current directory (`"."`). Typically used for Bookdown projects or R Markdown documents using bookdown output formats (e.g., `bookdown::pdf_document2`).
`verbose`	Logical. If `TRUE`, shows detailed line-by-line breakdown of all cross-references found. If `FALSE` (default), shows compact summary by file.

Details

This function helps users migrate from bookdown to Quarto by detecting cross-references that use bookdown syntax and need manual conversion.

Detected Cross-Reference Types:

Auto-detectable conversions:

- Figures: `\@ref(fig:label)` -> `@fig-label`
- Tables: `\@ref(tab:label)` -> `@tbl-label`
- Equations: `\@ref(eq:label)` -> `@eq-label`
- Sections: `\@ref(label)` -> `@sec-label`
- Theorems: `\@ref(thm:label)` -> `@thm-label` (also lem, cor, prp, def, exm, exr)

Manual conversion required:

- Numbered equations: `(\#eq:label)` -> requires equation restructuring
- Theorem blocks: Need explicit Quarto div syntax conversion All three formats from several bookdown versions are supported:
  - Old syntax with label: `{theorem, label="thm:label"}`
  - Old syntax without label: `{theorem chunk_name}`
  - New div syntax: `::: {.theorem #thm-label}`
- Section headers: Need explicit `{#sec-label}` IDs
- Figure labels: Need explicit `#| label: fig-label` in code chunks
- Table labels: Need explicit `#| label: tbl-label` in code chunks

Unsupported in Quarto:

- Conjecture (`cnj`) and Hypothesis (`hyp`) references

Adaptive Guidance:

The function provides **context-aware warnings** that only show syntax patterns actually found in your files. For example, if your project only uses the old theorem syntax without labels, you'll only see guidance for that specific pattern, not all possible variations.

Output Modes:

Default (`verbose = FALSE`)**:**

- Compact file-by-file summary
- Cross-reference counts by type
- Manual conversion requirements summary

Verbose (`verbose = TRUE`)**:**

- Detailed line-by-line breakdown
- Exact bookdown -> Quarto syntax transformations
- Context-aware conversion guidance showing only relevant syntax patterns
- Comprehensive examples with documentation links

Value

Invisibly returns a list of detected cross-references with their file locations, line numbers, and conversion details. Returns `NULL` if no cross-references are found.

See Also

Bookdown documentation:

- General: Bookdown book
- Cross-references
- Figures
- Tables
- Equations
- Theorems

Quarto documentation:

- Cross-references
- Cross-references with divs
- Figure cross-references
- Table cross-references

Examples

```
## Not run:
# Scan current directory (compact output)
detect_bookdown_crossrefs()

# Scan specific file with detailed output
detect_bookdown_crossrefs("my-document.Rmd", verbose = TRUE)

# Scan directory with context-aware guidance
detect_bookdown_crossrefs("path/to/bookdown/project", verbose = TRUE)

## End(Not run)
```

---

find_project_root *Find the root of a Quarto project*

Description

This function checks if the current working directory is within a Quarto project by looking for Quarto project files (`_quarto.yml` or `_quarto.yaml`). Unlike `get_running_project_root()`, this works both during rendering and interactive sessions.

Usage

```
find_project_root(path = ".")
```

Arguments

path	Character. Path to check for Quarto project files. Defaults to current working directory.

Value

Character Path of the project root directory if found, or NULL

See Also

get_running_project_root() for detecting active Quarto rendering

Examples

```
tmpdir <- tempfile()
dir.create(tmpdir)
find_project_root(tmpdir)
quarto_create_project("test-proj", type = "blog", dir = tmpdir, no_prompt = TRUE, quiet = TRUE)
blog_post_dir <- file.path(tmpdir, "test-proj", "posts", "welcome")
find_project_root(blog_post_dir)

xfun::in_dir(blog_post_dir, {
  # Check if current directory is a Quarto project or in one
  !is.null(find_project_root())
})

# clean up
unlink(tmpdir, recursive = TRUE)
```

---

get_running_project_root

Get the root of the currently running Quarto project

Description

This function is to be used inside cells and will return the project root when doing quarto_render() by detecting Quarto project environment variables.

Usage

```
get_running_project_root()
```

Details

Quarto sets `QUARTO_PROJECT_ROOT` and `QUARTO_PROJECT_DIR` environment variables when executing commands within a Quarto project context (e.g., `quarto render`, `quarto preview`). This function detects their presence.

Note that this function will return `NULL` when running code interactively in an IDE (even within a Quarto project directory), as these specific environment variables are only set during Quarto command execution.

Value

Character Quarto project root path from set environment variables.

Use in Quarto document cells

This function is particularly useful in Quarto document cells where you want to get the project root path dynamically during rendering. Cell example:

````
```{r}`r ''`
 # Get the project root path
 project_root <- get_running_project_root()
```
````

See Also

- `find_project_root()` for finding the Quarto project root directory
- `project_path()` for constructing paths relative to the project root

Examples

```
## Not run:
get_running_project_root()

## End(Not run)
```

---

has_parameters *Check if a Quarto document uses parameters*

Description

Determines whether a Quarto document uses parameters by examining the document structure and metadata. This function works with both knitr and Jupyter engines, using different detection methods for each:

Usage

```
has_parameters(input)
```

Arguments

`input`	Path to the Quarto document (.qmd or .ipynb file) to inspect.

Details

- **Knitr engine (.qmd files)**: Checks for a "params" field in the document's YAML metadata using `quarto_inspect()`
- **Jupyter engine (.ipynb files)**: Looks for code cells tagged with "parameters" following the papermill convention. For .ipynb files, the function parses the notebook JSON directly due to limitations in `quarto inspect`.

Parameters in Quarto enable creating dynamic, reusable documents. This function helps identify parameterized documents programmatically, which is useful for:

- Document processing workflows
- Automated report generation
- Parameter validation before rendering
- Project analysis and organization

For more information about using parameters in Quarto, see https://quarto.org/docs/computations/parameters.html

Value

Logical. `TRUE` if the document uses parameters, `FALSE` otherwise.

Examples

```
## Not run:
# Check if a document uses parameters
has_parameters("my-document.qmd")

# Check a parameterized report
has_parameters("parameterized-report.qmd")

# Check a Jupyter notebook
has_parameters("analysis.ipynb")

# Use in a workflow
if (has_parameters("report.qmd")) {
  message("This document accepts parameters")
}

## End(Not run)
```

is_using_quarto *Check is a directory is using quarto*

Description

This function will check if a directory is using quarto by looking for

- `_quarto.yml` at its root
- at least one `.qmd` file in the directory

Usage

```
is_using_quarto(dir = ".", verbose = FALSE)
```

Arguments

`dir`	The directory to check
`verbose`	print message about the result of the check

Examples

```
dir.create(tmpdir <- tempfile())
is_using_quarto(tmpdir)
file.create(file.path(tmpdir, "_quarto.yml"))
is_using_quarto(tmpdir)
unlink(tmpdir, recursive = TRUE)
```

new_blog_post *Create a new blog post*

Description

Creates (and potentially opens) the `index.qmd` file for a new blog post.

Usage

```
new_blog_post(
  title,
  dest = NULL,
  wd = NULL,
  open = rlang::is_interactive(),
  call = rlang::current_env(),
  ...
)
```

Arguments

title	A character string for the title of the post. It is converted to title case via tools::toTitleCase().
dest	A character string (or NULL) for the path within posts. By default, the title is adapted as the directory name.
wd	An optional working directory. If NULL, the current working is used.
open	A logical: have the default editor open a window to edit the index.qmd file?
call	The execution environment of a currently running function, e.g. caller_env(). The function will be mentioned in error messages as the source of the error. See the call argument of abort() for more information.
...	A named list of values to be added to the yaml header, such as date, author, categories, description, etc. If no date is provided, the current date is used. If no author is provided, whoami::fullname() is used to get the user's name.

Value

The path to the index file.

Examples

```
## Not run: \donttest{
new_blog_post("making quarto blog posts", categories = c("R"))
}
## End(Not run)
```

---

project_path *Get path relative to project root (Quarto-aware)*

Description

[Experimental]

This function constructs file paths relative to the project root when running in a Quarto context (using QUARTO_PROJECT_ROOT or QUARTO_PROJECT_DIR environment variables), or falls back to intelligent project root detection when not in a Quarto context.

It is experimental and subject to change in future releases. The automatic project root detection may not work reliably in all contexts, especially when projects have complex directory structures or when running in non-standard environments. For a more explicit and potentially more robust approach, consider using here::i_am() to declare your project structure, followed by here::here() for path construction. See examples for comparison.

Usage

```
project_path(..., root = NULL)
```

Arguments

`...`	Character vectors of path components to be joined
`root`	Project root directory. If `NULL` (default), automatic detection is used following the hierarchy described above

Details

The function uses the following fallback hierarchy to determine the project root:

- Quarto environment variables set during Quarto commands (e.g., `quarto render`):
  - `QUARTO_PROJECT_ROOT` environment variable (set by Quarto commands)
  - `QUARTO_PROJECT_DIR` environment variable (alternative Quarto variable)
- Fallback to intelligent project root detection using `xfun::proj_root()` for interactive sessions:
  - `_quarto.yml` or `_quarto.yaml` (Quarto project files)
  - `DESCRIPTION` file with `Package:` field (R package or Project)
  - `.Rproj` files with `Version:` field (RStudio projects)

Last fallback is the current working directory if no project root can be determined. A warning is issued to alert users that behavior may differ between interactive use and Quarto rendering, as in this case the computed path may be wrong.

Value

A character vector of the normalized file path relative to the project root.

Use in Quarto document cells

This function is particularly useful in Quarto document cells where you want to use a path relative to the project root dynamically during rendering.

````
```{r}`r ''`
 # Get a csv path from data directory in the Quarto project root
 data <- project_path("data", "my_data.csv")
```
````

See Also

- `here::here()` and `here::i_am()` for a similar function that works with R projects
- `find_project_root()` to search for Quarto Project configuration in parents directories
- `get_running_project_root()` for detecting the project root in Quarto commands
- `xfun::from_root()` for the underlying path construction
- `xfun::proj_root()` for project root detection logic

Examples

```
## Not run:
# Create a dummy Quarto project structure for example
tmpdir <- tempfile("quarto_project")
dir.create(tmpdir)
quarto::quarto_create_project(
  'test project', type = 'blog',
  dir = tmpdir, no_prompt = TRUE, quiet = TRUE
)
project_dir <- file.path(tmpdir, "test project")

# Simulate working within a blog post
xfun::in_dir(
  dir = file.path(project_dir, "posts", "welcome"), {

  # Reference a data file from project root
  # ../../data/my_data.csv
  quarto::project_path("data", "my_data.csv")

  # Reference a script from project root
  # ../../R/analysis.R
  quarto::project_path("R", "analysis.R")

  # Explicitly specify root (overrides automatic detection)
  # ../../data/file.csv
  quarto::project_path("data", "file.csv", root = "../..")

  # Alternative approach using here::i_am() (potentially more robust)
  # This approach requires you to declare where you are in the project:
  if (requireNamespace("here", quietly = TRUE)) {
    # Declare that this document is in the project root or subdirectory
    here::i_am("posts/welcome/index.qmd")

    # Now here::here() will work reliably from the project root
    here::here("data", "my_data.csv")
    here::here("R", "analysis.R")
  }
})


## End(Not run)
```

`qmd_to_r_script` *Convert Quarto document to R script*

Description

[Experimental]

Extracts R code cells from a Quarto document and writes them to an R script file that can be rendered with the same options. The Markdown text is not preserved, but R chunk options are kept as comment headers using Quarto's `#|` syntax.

This function is still experimental and may slightly change in future releases, depending on feedback.

Usage

```
qmd_to_r_script(qmd, script = NULL)
```

Arguments

`qmd`	Character. Path to the input Quarto document (.qmd file).
`script`	Character. Path to the output R script file. If `NULL` (default), the script file will have the same name as the input file but with `.R` extension.

Details

This function processes a Quarto document by:

- Extracting only R code cells (markdown and cell in other languages are ignored)
- Preserving chunk options as `#|` comment headers
- Adding the document's YAML metadata as a spin-style header
- Creating an R script that can be rendered with the same options

Chunk option handling::

- Chunks with `purl: false` are completely skipped and not included in the output
- Chunks with `eval: false` have their code commented out (prefixed with `# `) in the R script
- All other chunk options are preserved as `#|` comment headers

File handling::

- If the output R script already exists, the function will abort with an error
- Non-R code cells (e.g., Python, Julia, Observable JS) are ignored
- If no R code cells are found, the function does nothing and returns `NULL`

Compatibility::

The resulting R script is compatible with Quarto's script rendering via `knitr::spin()` and can be rendered directly with `quarto render script.R`. See https://quarto.org/docs/computations/render-scripts.html#knitr for more details on rendering R scripts with Quarto.

The resulting R script uses Quarto's executable cell format with `#|` comments to preserve chunk options like `label`, `echo`, `output`, etc.

The resulting R script could also be `source()`d in R, as any `eval = FALSE` will be commented out.

Limitations::

This function relies on static analysis of the Quarto document by `quarto inspect`. This means that any **knitr** specific options like `child=` or specific feature like `knitr::read_chunk()` are not supported. They rely on tangling or knitting by **knitr** itself. For this support, one should look at `knitr::hook_purl()` or `knitr::purl()`.

Value

Invisibly returns the path to the created R script file, or NULL if no R code cells were found.

Examples

```
## Not run:
# Convert a Quarto document to R script
qmd_to_r_script("my-document.qmd")
# Creates "my-document.R"

# Specify custom output file
qmd_to_r_script("my-document.qmd", script = "extracted-code.R")

## End(Not run)
```

---

quarto_add_extension *Install a Quarto extensions*

Description

Add an extension to this folder or project by running `quarto add`

Usage

```
quarto_add_extension(
  extension = NULL,
  no_prompt = FALSE,
  quiet = FALSE,
  quarto_args = NULL
)
```

Arguments

extension	The extension to install, either an archive or a GitHub repository as described in the documentation https://quarto.org/docs/extensions/managing.html.
no_prompt	Do not prompt to confirm approval to download external extension.
quiet	Suppress warning and other messages, from R and also Quarto CLI (i.e `--quiet` is passed as command line). `quarto.quiet` R option or `R_QUARTO_QUIET` environment variable can be used to globally override a function call (This can be useful to debug tool that calls `quarto_*` functions directly). On Github Actions, it will always be `quiet = FALSE`.
quarto_args	Character vector of other `quarto` CLI arguments to append to the Quarto command executed by this function. This is mainly intended for advanced usage and useful for CLI arguments which are not yet mirrored in a dedicated parameter of this R function. See `quarto render --help` for options.

Extension Trust

Quarto extensions may execute code when documents are rendered. Therefore, if you do not trust the author of an extension, we recommend that you do not install or use the extension. By default `no_prompt = FALSE` which means that the function will ask for explicit approval when used interactively, or disallow installation.

Examples

```
## Not run:
# Install a template and set up a draft document from a GitHub repository
quarto_add_extension("quarto-ext/fontawesome")

# Install a template and set up a draft document from a ZIP archive
quarto_add_extension("https://github.com/quarto-ext/fontawesome/archive/refs/heads/main.zip")

## End(Not run)
```

`quarto_available` *Check if quarto is available and version meet some requirements*

Description

This function allows to test if Quarto is available and meets version requirement, a min, max or in between requirement.

Usage

```
quarto_available(min = NULL, max = NULL, error = FALSE)
```

Arguments

`min`	Minimum version expected.
`max`	Maximum version expected
`error`	If `TRUE`, will throw an error if Quarto is not available or does not meet the requirement. Default is `FALSE`.

Details

If `min` and `max` are provided, this will check if Quarto version is in-between two versions. If non is provided (keeping the default `NULL` for both), it will just check for Quarto availability version and return `FALSE` if not found.

Value

logical. `TRUE` if requirement is met, `FALSE` otherwise.

Examples

```
# Is there an active version available ?
quarto_available()
# check for a minimum requirement
quarto_available(min = "1.5")
# check for a maximum version
quarto_available(max = "1.6")
# only returns TRUE if Pandoc version is between two bounds
quarto_available(min = "1.4", max = "1.6")
```

`quarto_binary_sitrep` *Check configurations for quarto binary used*

Description

This function check the configuration for the quarto package R package to detect a possible difference with version used by RStudio IDE.

Usage

```
quarto_binary_sitrep(verbose = TRUE, debug = FALSE)
```

Arguments

`verbose`	if `FALSE`, only return the result of the check.
`debug`	if `TRUE`, print more information about value set in configurations.

Value

`TRUE` if this package should be using the same quarto binary as the RStudio IDE. `FALSE` otherwise if a difference is detected or quarto is not found. Use `verbose = TRUE` or`debug = TRUE` to get detailed information.

Examples

```
quarto_binary_sitrep(verbose = FALSE)
quarto_binary_sitrep(verbose = TRUE)
quarto_binary_sitrep(debug = TRUE)
```

quarto_create_project *Create a quarto project*

Description

This function calls `quarto create project <type> <name>`. It creates a new directory with the project name, inside the requested parent directory, and adds some starter files that are appropriate to the project type.

Usage

```
quarto_create_project(
  name,
  type = "default",
  dir = ".",
  title = name,
  no_prompt = FALSE,
  quiet = FALSE,
  quarto_args = NULL
)
```

Arguments

`name`	The name of the project and the directory that will be created. Special case is to use `name = "."` to create the project in the current directory. In that case provide `title` to set the project title.
`type`	The type of project to create. As of Quarto 1.4, it can be one of `default`, `website`, `blog`, `book`, `manuscript`, `confluence`.
`dir`	The directory in which to create the new Quarto project, i.e. the parent directory.
`title`	The title of the project. By default, it will be the name of the project, same as directory name created. or "My project" if `name = "."`. If you want to set a different title, provide it here.
`no_prompt`	Do not prompt to approve the creation of the new project folder.
`quiet`	Suppress warning and other messages, from R and also Quarto CLI (i.e `--quiet` is passed as command line). `quarto.quiet` R option or `R_QUARTO_QUIET` environment variable can be used to globally override a function call (This can be useful to debug tool that calls `quarto_*` functions directly). On Github Actions, it will always be `quiet = FALSE`.
`quarto_args`	Character vector of other `quarto` CLI arguments to append to the Quarto command executed by this function. This is mainly intended for advanced usage and useful for CLI arguments which are not yet mirrored in a dedicated parameter of this R function. See `quarto render --help` for options.

Quarto version required

This function requires Quarto 1.4 or higher. Use `quarto_version()` to see your current Quarto version.

See Also

Quarto documentation on Quarto projects

Examples

```
## Not run:
# Create a new project directory in another directory
quarto_create_project("my-first-quarto-project", dir = "~/tmp")

# Create a new project directory in the current directory
quarto_create_project("my-first-quarto-project")

# Create a new project with a different title
quarto_create_project("my-first-quarto-project", title = "My Quarto Project")

# Create a new project inside the current directory directly
quarto_create_project(".", title = "My Quarto Project")

## End(Not run)
```

---

quarto_inspect *Inspect Quarto Input File or Project*

Description

Inspect a Quarto project or input path. Inspecting a project returns its config and engines. Inspecting an input path return its formats, engine, and dependent resources.

Usage

```
quarto_inspect(input = ".", profile = NULL, quiet = FALSE, quarto_args = NULL)
```

Arguments

`input`	The input file or project directory to inspect.
`profile`	Quarto project profile(s) to use. Either a character vector of profile names or `NULL` to use the default profile.
`quiet`	Suppress warning and other messages, from R and also Quarto CLI (i.e `--quiet` is passed as command line).

quarto.quiet R option or R_QUARTO_QUIET environment variable can be used to globally override a function call (This can be useful to debug tool that calls quarto_* functions directly).

On Github Actions, it will always be quiet = FALSE.

quarto_args — Character vector of other quarto CLI arguments to append to the Quarto command executed by this function. This is mainly intended for advanced usage and useful for CLI arguments which are not yet mirrored in a dedicated parameter of this R function. See quarto render --help for options.

Value

Named list. For input files, the list contains the elements quarto, engines, formats, resources, fileInformation plus project if the file is part of a Quarto project. For projects, the list contains the elements quarto, dir, engines, config and files.

Examples

```
## Not run:
# Inspect input file file
quarto_inspect("notebook.Rmd")

# Inspect project
quarto_inspect("myproject")

# Inspect project's advanced profile
quarto_inspect(
  input = "myproject",
  profile = "advanced"
)

## End(Not run)
```

---

quarto_list_extensions

List Installed Quarto extensions

Description

List Quarto Extensions in this folder or project by running quarto list

Usage

```
quarto_list_extensions()
```

Value

A data frame with the installed extensions or NULL (invisibly) if no extensions are installed.

Examples

```
## Not run:
# List Quarto Extensions in this folder or project
quarto_list_extensions()

## End(Not run)
```

`quarto_path` *Path to the quarto binary*

Description

Determine the path to the quarto binary. Uses `QUARTO_PATH` environment variable if defined, otherwise uses `Sys.which()`.

Usage

```
quarto_path(normalize = TRUE)
```

Arguments

`normalize`	If `TRUE` (default), normalize the path using `base::normalizePath()`.

Value

Path to quarto binary (or `NULL` if not found)

See Also

`quarto_version()` to check the version of the binary found, `quarto_available()` to check if Quarto CLI is available and meets some requirements.

`quarto_preview` *Quarto Preview*

Description

Render and preview a Quarto document or website project.

Usage

```
quarto_preview(
  file = NULL,
  render = "auto",
  port = "auto",
  host = "127.0.0.1",
  browse = TRUE,
  watch = TRUE,
  navigate = TRUE,
  quiet = FALSE
)

quarto_preview_stop()
```

Arguments

file	The document or website project directory to preview (defaults to current working directory)
render	For website preview, the most recent execution results of computational documents are used to render the site (this is to optimize startup time). If you want to perform a full render prior to serving pass "all" or a vector of specific formats to render. Pass "default" to render the default format for the site. For document preview, the document is rendered prior to preview (pass FALSE to override this).
port	Port to listen on (defaults to 4848)
host	Hostname to bind to (defaults to 127.0.0.1)
browse	Open a browser to preview the content. Defaults to using the RStudio Viewer when running within RStudio.Pass a function (e.g. utils::browseURL to override this behavior).
watch	Watch for changes and automatically reload browser.
navigate	Automatically navigate the preview browser to the most recently rendered document.
quiet	Suppress warning and other messages, from R and also Quarto CLI (i.e --quiet is passed as command line)

Details

Automatically reloads the browser when input files are re-rendered or document resources (e.g. CSS) change.

Value

The URL of the preview server (invisibly). This can be used to programmatically access the server location, for example to take screenshots with webshot2 or pass to other automation tools.

Examples

```
## Not run:
# Preview the project in the current directory
quarto_preview()

# Preview a document
quarto_preview("document.qmd")

# Preview the project in "myproj" directory and use external browser
# (rather than RStudio Viewer)
quarto_preview("myproj", open = utils::browseURL)

# Capture the preview URL for programmatic use
preview_url <- quarto_preview("document.qmd", browse = FALSE)
cat("Preview available at:", preview_url, "\n")

# Take a screenshot of the preview using webshot2
if (require(webshot2)) {
  preview_url <- quarto_preview("document.qmd", browse = FALSE)
  webshot2::webshot(preview_url, "preview.png")
}

# Stop any running quarto preview
quarto_preview_stop()

## End(Not run)
```

---

quarto_publish_doc *Publish Quarto Documents*

Description

Publish Quarto documents to Posit Connect, ShinyApps, and RPubs

Usage

```
quarto_publish_doc(
  input,
  name = NULL,
  title = NULL,
  server = NULL,
  account = NULL,
  render = c("local", "server", "none"),
  metadata = list(),
  ...
)

quarto_publish_app(
```

```
  input = getwd(),
  name = NULL,
  title = NULL,
  server = NULL,
  account = NULL,
  render = c("local", "server", "none"),
  metadata = list(),
  ...
)

quarto_publish_site(
  input = getwd(),
  name = NULL,
  title = NULL,
  server = NULL,
  account = NULL,
  render = c("local", "server", "none"),
  metadata = list(),
  ...
)
```

Arguments

`input`	The input file or project directory to be published. Defaults to the current working directory.
`name`	Name for publishing (names must be unique within an account). Defaults to the name of the `input`.
`title`	Free-form descriptive title of application. Optional; if supplied, will often be displayed in favor of the name. When deploying a new document, you may supply only the title to receive an auto-generated name
`account`, `server`	Uniquely identify a remote server with either your user `account`, the `server` name, or both. If neither are supplied, and there are multiple options, you'll be prompted to pick one. Use `accounts()` to see the full list of available options.
`render`	`local` to render locally before publishing; `server` to render on the server; `none` to use whatever rendered content currently exists locally. (defaults to `local`)
`metadata`	Additional metadata fields to save with the deployment record. These fields will be returned on subsequent calls to `deployments()`. Multi-value fields are recorded as comma-separated values and returned in that form. Custom value serialization is the responsibility of the caller.
`...`	Named parameters to pass along to `rsconnect::deployApp()`

Examples

```
## Not run:
library(quarto)
quarto_publish_doc("mydoc.qmd")
```

```
quarto_publish_app(server = "shinyapps.io")
quarto_publish_site(server = "rstudioconnect.example.com")

## End(Not run)
```

`quarto_remove_extension` *Remove a Quarto extensions*

Description

Remove an extension in this folder or project by running `quarto remove`

Usage

```
quarto_remove_extension(
  extension = NULL,
  no_prompt = FALSE,
  quiet = FALSE,
  quarto_args = NULL
)
```

Arguments

`extension`	The extension name to remove, as in `quarto remove <extension-name>`.
`no_prompt`	Do not prompt to confirm approval to download external extension.
`quiet`	Suppress warning and other messages, from R and also Quarto CLI (i.e `--quiet` is passed as command line). `quarto.quiet` R option or `R_QUARTO_QUIET` environment variable can be used to globally override a function call (This can be useful to debug tool that calls `quarto_*` functions directly). On Github Actions, it will always be `quiet = FALSE`.
`quarto_args`	Character vector of other `quarto` CLI arguments to append to the Quarto command executed by this function. This is mainly intended for advanced usage and useful for CLI arguments which are not yet mirrored in a dedicated parameter of this R function. See `quarto render --help` for options.

Value

Returns invisibly `TRUE` if the extension was removed, `FALSE` otherwise.

See Also

`quarto_add_extension()` and Quarto Website.

Examples

```
## Not run:
# Remove an already installed extension
quarto_remove_extension("quarto-ext/fontawesome")

## End(Not run)
```

`quarto_render` *Render Markdown*

Description

Render the input file to the specified output format using quarto. If the input requires computations (e.g. for Rmd or Jupyter files) then those computations are performed before rendering.

Usage

```
quarto_render(
  input = NULL,
  output_format = NULL,
  output_file = NULL,
  execute = TRUE,
  execute_params = NULL,
  execute_dir = NULL,
  execute_daemon = NULL,
  execute_daemon_restart = FALSE,
  execute_debug = FALSE,
  use_freezer = FALSE,
  cache = NULL,
  cache_refresh = FALSE,
  metadata = NULL,
  metadata_file = NULL,
  debug = FALSE,
  quiet = FALSE,
  profile = NULL,
  quarto_args = NULL,
  pandoc_args = NULL,
  as_job = getOption("quarto.render_as_job", "auto")
)
```

Arguments

`input`	The input file or project directory to be rendered (defaults to rendering the project in the current working directory).
`output_format`	Target output format (defaults to `"html"`). The option `"all"` will render all formats defined within the file or project.

output_file	Base name for single-file output (e.g. PDF, ePub, MS Word). This sets the output-file Quarto metadata. If NULL, the output filename will be based on the input filename.
execute	Whether to execute embedded code chunks.
execute_params	A list of named parameters that override custom params specified within the YAML front-matter.
execute_dir	The working directory in which to execute embedded code chunks.
execute_daemon	Keep Jupyter kernel alive (defaults to 300 seconds). Note this option is only applicable for rendering Jupyter notebooks or Jupyter markdown.
execute_daemon_restart	Restart keepalive Jupyter kernel before render. Note this option is only applicable for rendering Jupyter notebooks or Jupyter markdown.
execute_debug	Show debug output for Jupyter kernel.
use_freezer	Force use of frozen computations for an incremental file render.
cache	Cache execution output (uses knitr cache and jupyter-cache respectively for Rmd and Jupyter input files).
cache_refresh	Force refresh of execution cache.
metadata	An optional named list used to override YAML metadata. It will be passed as a YAML file to --metadata-file CLI flag. This will be merged over metadata-file options if both are specified.
metadata_file	A yaml file passed to --metadata-file CLI flags to override metadata. This will be merged with metadata if both are specified, with low precedence on metadata options.
debug	Leave intermediate files in place after render.
quiet	Suppress warning and other messages, from R and also Quarto CLI (i.e --quiet is passed as command line). quarto.quiet R option or R_QUARTO_QUIET environment variable can be used to globally override a function call (This can be useful to debug tool that calls quarto_* functions directly). On Github Actions, it will always be quiet = FALSE.
profile	Quarto project profile(s) to use. Either a character vector of profile names or NULL to use the default profile.
quarto_args	Character vector of other quarto CLI arguments to append to the Quarto command executed by this function. This is mainly intended for advanced usage and useful for CLI arguments which are not yet mirrored in a dedicated parameter of this R function. See quarto render --help for options.
pandoc_args	Additional command line arguments to pass on to Pandoc.
as_job	Render as an RStudio background job. Default is "auto", which will render individual documents normally and projects as background jobs. Use the quarto.render_as_job R option to control the default globally.

Value

Invisibly returns NULL. The function is called for its side effect of rendering the specified document or project.

Examples

```
## Not run:
# Render R Markdown
quarto_render("notebook.Rmd")
quarto_render("notebook.Rmd", output_format = "pdf")

# Render Jupyter Notebook
quarto_render("notebook.ipynb")

# Render Jupyter Markdown
quarto_render("notebook.md")

# Override metadata
quarto_render("notebook.Rmd", metadata = list(lang = "fr", execute = list(echo = FALSE)))

## End(Not run)
```

---

quarto_serve *Serve Interactive Document*

Description

Serve a Shiny interactive document. By default, the document will be rendered first and then served If you have previously rendered the document, pass `render = FALSE` to skip rendering.

Usage

```
quarto_serve(
  input,
  render = TRUE,
  port = getOption("shiny.port"),
  host = getOption("shiny.host", "127.0.0.1"),
  browse = TRUE
)
```

Arguments

`input`	The input file to run. Should be a file with a `server: shiny` entry in its YAML front-matter.
`render`	Render the document before serving it.
`port`	Port to listen on (defaults to 4848)
`host`	Hostname to bind to (defaults to 127.0.0.1)
`browse`	Open a browser to preview the content. Defaults to using the RStudio Viewer when running within RStudio.Pass a function (e.g. `utils::browseURL` to override this behavior).

`quarto_update_extension` *Update a Quarto extensions*

Description

Update an extension to this folder or project by running `quarto update`

Usage

```
quarto_update_extension(
  extension = NULL,
  no_prompt = FALSE,
  quiet = FALSE,
  quarto_args = NULL
)
```

Arguments

`extension`	The extension to update, either by its name (i.e `quarto update extension <gh-org>/<gh-repo>`), an archive (`quarto update extension <path-to-zip>`) or a url (`quarto update extension <url>`).
`no_prompt`	Do not prompt to confirm approval to download external extension. Setting `no_prompt = FALSE` means Extension Trust is accepted.
`quiet`	Suppress warning and other messages, from R and also Quarto CLI (i.e `--quiet` is passed as command line). `quarto.quiet` R option or `R_QUARTO_QUIET` environment variable can be used to globally override a function call (This can be useful to debug tool that calls `quarto_*` functions directly). On Github Actions, it will always be `quiet = FALSE`.
`quarto_args`	Character vector of other `quarto` CLI arguments to append to the Quarto command executed by this function. This is mainly intended for advanced usage and useful for CLI arguments which are not yet mirrored in a dedicated parameter of this R function. See `quarto render --help` for options.

Value

Returns invisibly `TRUE` if the extension was updated, `FALSE` otherwise.

Extension Trust

Quarto extensions may execute code when documents are rendered. Therefore, if you do not trust the author of an extension, we recommend that you do not install or use the extension. By default `no_prompt = FALSE` which means that the function will ask for explicit approval when used interactively, or disallow installation.

See Also

quarto_add_extension(), quarto_remove_extension(), and Quarto website.

Examples

```
## Not run:
# Update a template and set up a draft document from a GitHub repository
quarto_update_extension("quarto-ext/fontawesome")

# Update a template and set up a draft document from a ZIP archive
quarto_update_extension("https://github.com/quarto-ext/fontawesome/archive/refs/heads/main.zip")

## End(Not run)
```

---

quarto_use_template *Use a custom format extension template*

Description

Install and use a template for Quarto using `quarto use`.

Usage

```
quarto_use_template(
  template,
  dir = ".",
  no_prompt = FALSE,
  quiet = FALSE,
  quarto_args = NULL
)
```

Arguments

`template`	The template to install, either an archive or a GitHub repository as described in the documentation https://quarto.org/docs/extensions/formats.html.
`dir`	The directory in which to install the template. This must be an empty directory. To use directly in a non-empty directory, use `quarto use template` interactively in the terminal for safe installation without overwrite.
`no_prompt`	Do not prompt to confirm approval to download external extension.
`quiet`	Suppress warnings and messages.
`quarto_args`	Character vector of other `quarto` CLI arguments to append to the Quarto command executed by this function. This is mainly intended for advanced usage and useful for CLI arguments which are not yet mirrored in a dedicated parameter of this R function. See `quarto render --help` for options.

Examples

```
## Not run:
# Use a template and set up a draft document from a GitHub repository
quarto_use_template("quarto-journals/jss")

# Use a template in current directory by installing it in an empty directory
quarto_use_template("quarto-journals/jss", dir = "new-empty-dir")

# Use a template and set up a draft document from a ZIP archive
quarto_use_template("https://github.com/quarto-journals/jss/archive/refs/heads/main.zip")

## End(Not run)
```

---

quarto_version *Check quarto version*

Description

Determine the specific version of quarto binary found by `quarto_path()`. If it returns `99.9.9` then it means you are using a dev version.

Usage

```
quarto_version()
```

Value

a `numeric_version` with the quarto version found

See Also

`quarto_available()` to check if the version meets some requirements.

---

tbl_qmd_elements *Create Quarto Markdown HTML Elements for Tables*

Description

Functions to wrap content in HTML spans or divs with data-qmd attributes for Quarto processing within HTML tables. These functions are specifically designed for use with HTML table packages like kableExtra, gt, or DT where you need Quarto to process markdown content within table cells.

Usage

```
tbl_qmd_span(content, display = NULL, use_base64 = TRUE)

tbl_qmd_div(content, display = NULL, use_base64 = TRUE)

tbl_qmd_span_base64(content, display = NULL)

tbl_qmd_div_base64(content, display = NULL)

tbl_qmd_span_raw(content, display = NULL)

tbl_qmd_div_raw(content, display = NULL)
```

Arguments

content	Character string of content to wrap. This can include Markdown, LaTeX math, and Quarto shortcodes.
display	Optional display text (if different from content). Useful for fallback text when Quarto processing is not available or for better accessibility.
use_base64	Logical, whether to base64 encode the content (recommended for complex content with special characters or when content includes quotes)

Details

These functions create HTML elements with `data-qmd` or `data-qmd-base64` attributes that Quarto processes during document rendering. The base64 encoding is recommended for content with special characters, quotes, or complex formatting.

Available functions:

- `tbl_qmd_span()` and `tbl_qmd_div()` are the main functions with encoding options
- `tbl_qmd_span_base64()` and `tbl_qmd_div_base64()` explicitly use base64 encoding
- `tbl_qmd_span_raw()` and `tbl_qmd_div_raw()` explicitly use raw encoding

This feature requires Quarto version 1.3 or higher with HTML format outputs. For more information, see https://quarto.org/docs/authoring/tables.html#html-tables.

Value

Character string containing the HTML element with appropriate data-qmd attributes

Examples

```
# Basic span usage in table cells
tbl_qmd_span("**bold text**")
tbl_qmd_span("$\\alpha + \\beta$", display = "Greek formula")

# Basic div usage in table cells
tbl_qmd_div("## Section Title\n\nContent here")
```

```
tbl_qmd_div("{{< video https://example.com >}}", display = "[Video content]")

# Explicit encoding choices
tbl_qmd_span_base64("Complex $\\LaTeX$ content")
tbl_qmd_span_raw("Simple text")

# Use with different HTML table packages
## Not run:
# With kableExtra
library(kableExtra)
df <- data.frame(
  math = c(tbl_qmd_span("$x^2$"), tbl_qmd_span("$\\sum_{i=1}^n x_i$")),
  text = c(tbl_qmd_span("**Important**", "bold"), tbl_qmd_span("`code`", "code"))
)
kbl(df, format = "html", escape = FALSE) |> kable_styling()

## End(Not run)
```

`theme_colors_flextable` *Create a theme for a plotting or table package*

Description

Create a theme using background and foreground colors (`theme_colors_*`) or using a **brand.yml** file (`theme_brand_*`).

Usage

```
theme_colors_flextable(bg, fg)

theme_brand_flextable(brand_yml)

theme_colors_ggplot2(bg, fg)

theme_brand_ggplot2(brand_yml)

theme_colors_gt(bg, fg)

theme_brand_gt(brand_yml)

theme_colors_plotly(bg, fg)

theme_brand_plotly(brand_yml)

theme_colors_thematic(bg, fg)

theme_brand_thematic(brand_yml)
```

Arguments

bg	The background color
fg	The foreground color
brand_yml	The path to a brand.yml file

Details

The use of the theme will depend on the package. Please see light/dark renderings examples for examples using each supported package with dark mode, `theme_brand_*`, and `renderings: [light, dark]`, or theme helper article for examples using each package and `theme_colors_*` to specify the background and foreground colors directly.

---

write_yaml_metadata_block

Write YAML Metadata Block for Quarto Documents

Description

Creates a YAML metadata block that can be dynamically inserted into Quarto documents from R code chunks. This allows setting metadata values based on R computations, which can then be used with Quarto's conditional content features like `when-meta` and `{{< meta >}}` shortcodes.

Usage

```
write_yaml_metadata_block(..., .list = NULL)
```

Arguments

...	Named arguments to include in the metadata block. Names become the metadata keys and values become the metadata values. These take precedence over any conflicting keys in `.list`.
.list	Optional list of additional metadata to include. This is useful when you have metadata stored in a list variable. Keys in `.list` are overridden by any matching keys provided in `...`.

Details

The function converts R values to YAML format and wraps them in YAML delimiters (`---`). Logical values are converted to lowercase strings ("true"/"false") to ensure compatibility with Quarto's metadata system.

When both `...` and `.list` contain the same key, the value from `...` takes precedence and will override the value from `.list`.

If no metadata is provided (empty `...` and `NULL` or empty `.list`), the function returns `NULL` without generating any output.

This addresses the limitation where Quarto metadata must be static and cannot be set dynamically from R code during document rendering.

YAML 1.2 Compatibility::

To ensure compatibility with Quarto's YAML 1.2 parser (js-yaml), the function automatically handles two key differences between R's yaml package (YAML 1.1) and YAML 1.2:

Boolean values::
R logical values (`TRUE`/`FALSE`) are converted to lowercase YAML 1.2 format (`true`/`false`) using [yaml::verbatim_logical()]. This prevents YAML 1.1 boolean representations like `yes`/`no` from being used.

String quoting::
Strings with leading zeros that contain digits 8 or 9 (like `"029"`, `"089"`) are automatically quoted to prevent them from being parsed as octal numbers, which would result in data corruption (e.g., `"029"` becoming `29`). Valid octal numbers containing only digits 0-7 (like `"0123"`) are handled by the underlying **yaml** package.
For manual control over string quoting behavior, use [yaml_quote_string()].

Quarto Usage::

To use this function in a Quarto document, create an R code chunk with the `output: asis` option:

```
```{r}
#| output: asis
write_yaml_metadata_block(admin = TRUE, version = "1.0")
```

Without the `output: asis` option, the YAML metadata block will be
displayed as text rather than processed as metadata by Quarto.

[yaml::verbatim_logical()]: R:yaml::verbatim_logical()
[yaml_quote_string()]: R:yaml_quote_string()
```

Value

A character string containing the formatted YAML metadata block, wrapped with `knitr::asis_output()` so it renders as raw markdown. Returns `NULL` invisibly if no metadata is provided.

See Also

`yaml_quote_string()` for explicitly controlling which strings are quoted in YAML output when you encounter edge cases that need manual handling.

Examples

```
## Not run:
# In a Quarto document R chunk with `#| output: asis`:
admin <- TRUE
user_level <- "advanced"

# Set metadata dynamically
write_yaml_metadata_block(
  admin = admin,
  level = user_level,
  timestamp = Sys.Date()
```

```
)

# Strings with leading zeros are automatically quoted for YAML 1.2 compatibility
write_yaml_metadata_block(
  zip_code = "029",    # Automatically quoted as "029"
  build_id = "0123"    # Quoted by yaml package (valid octal)
)

# Use with .list parameter
metadata_list <- list(version = "1.0", debug = FALSE)
write_yaml_metadata_block(.list = metadata_list)

# Direct arguments override .list values
base_config <- list(theme = "dark", debug = TRUE)
write_yaml_metadata_block(
  debug = FALSE,  # This overrides debug = TRUE from base_config
  author = "John",
  .list = base_config
)

# Then use in Quarto with conditional content:
# ::: {.content-visible when-meta="admin"}
# Admin-only content here
# :::

## End(Not run)
```

`yaml_quote_string` *Add quoted attribute to strings for YAML output*

Description

This function allows users to explicitly mark strings that should be quoted in YAML output, giving full control over quoting behavior.

Usage

```
yaml_quote_string(x)
```

Arguments

x	A character vector or single string

Details

This is particularly useful for special values that might be misinterpreted as **yaml** uses YAML 1.1 and Quarto expects YAML 1.2.

The `quoted` attribute is a convention used by `yaml::as.yaml()`

Value

The input with quoted attributes applied

Examples

```
yaml::as.yaml(list(id = yaml_quote_string("1.0")))
yaml::as.yaml(list(id = "1.0"))
```

Index