Package ‘froggeR’

November 14, 2025

Type Package

Title Enhance 'Quarto' Project Workflows and Standards

Version 0.6.0

Maintainer Kyle Grealis <kyleGrealis@icloud.com>

Description Streamlines 'Quarto' workflows by providing tools for consistent project setup and documentation. Enables portability through reusable metadata, automated project structure creation, and standardized templates. Features include enhanced project initialization, pre-formatted 'Quarto' documents, inclusion of 'Quarto' brand functionality, comprehensive data protection settings, custom styling, and structured documentation generation. Designed to improve efficiency and collaboration in R data science projects by reducing repetitive setup tasks while maintaining consistent formatting across multiple documents.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 3.5.0)

Imports cli (>= 3.0.0), fs, glue (>= 1.6.0), here (>= 1.0.1), quarto (>= 1.3.0), rappdirs, readr (>= 2.0.0), rlang, rstudioapi, stringr (>= 1.5.0), usethis (>= 2.2.0), yaml

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

Config/build/copy-resources inst/gists/basic_quarto.qmd inst/gists/custom_quarto.qmd inst/gists/README.md

Config/testthat/edition 3

VignetteBuilder knitr

URL https://www.kyleGrealis.com/froggeR/

BugReports https://github.com/kyleGrealis/froggeR/issues

NeedsCompilation no

Author Kyle Grealis [aut, cre] (ORCID: <https://orcid.org/0000-0002-9223-8854>)

Repository CRAN

Date/Publication 2025-11-14 20:00:09 UTC

Contents

`brand_settings` *Manage Brand Settings for froggeR Quarto Projects*

Description

Interactively configure your branding (colors, logos, organization name) for reuse across multiple Quarto projects.

Usage

```
brand_settings()
```

Details

This function provides an interactive workflow with three main options:

1. **Create or update branding**: Interactively configure:
   - Organization/project name
   - Primary brand color (hex format, e.g., #0066cc)
   - Large logo path
   - Small/icon logo path
   - Optional: Advanced customization through template editing
2. **View current branding**: Display existing project-level and global brand configurations
3. **Exit**: Return to console

After configuring basic settings, you can choose to:

- Save with the basics (quick setup)
- Edit the full template for advanced options (typography, color palettes, links, code styling, headings)

Branding is stored as YAML (`_brand.yml`) in:

- Project-level: `_brand.yml` in the current Quarto project
- Global: System-wide config directory (reused across all projects)
- Both: Saved to both locations

Global branding is automatically applied in new projects via `quarto_project` and `write_brand`.

Value

Invisibly returns `NULL`. Called for side effects (interactive configuration).

See Also

`settings`, `write_brand`, `save_brand`

Examples

```
# Open interactive brand settings configuration
if (interactive()) {
  froggeR::brand_settings()
}
```

`quarto_project` *Create a Custom Quarto Project*

Description

This function creates a new Quarto project directory with additional froggeR features. It first calls `quarto::quarto_create_project()` to set up the basic structure, then enhances it with froggeR-specific files and settings.

Usage

```
quarto_project(name, path = here::here(), example = TRUE)
```

Arguments

`name`	Character. The name of the Quarto project directory and initial `.qmd` file. Must contain only letters, numbers, hyphens, and underscores.
`path`	Character. Path to the parent directory where project will be created. Default is current project root via `here`.
`example`	Logical. If `TRUE` (default), creates a Quarto document with brand logo positioning and examples of within-document cross-referencing, links, and references.

Details

This function creates a Quarto project with the following enhancements:

- `_brand.yml`: Stores Quarto project branding
- `logos`: Quarto project branding logos directory
- `_variables.yml`: Stores reusable YAML variables
- `.gitignore`: Enhanced settings for R projects
- `README.md`: Template README file
- `dated_progress_notes.md`: For project progress tracking
- `custom.scss`: Custom Quarto styling
- `references.bib`: Bibliography template

The function requires Quarto version 1.6 or greater. Global froggeR settings are automatically applied if available.

Value

Invisibly returns the path to the created project directory.

See Also

`write_quarto`, `settings`, `brand_settings`, `write_brand`, `write_variables`, `save_brand`, `save_variables`

Examples

```
if (interactive() && quarto::quarto_version() >= "1.6") {

  # Create the Quarto project with custom YAML & associated files
  quarto_project("frogs", path = tempdir(), example = TRUE)

  # Confirm files were created
  file.exists(file.path(tempdir(), "frogs", "frogs.qmd"))
  file.exists(file.path(tempdir(), "frogs", "_quarto.yml"))

}
```

`save_brand` *Save Brand Configuration to Global froggeR Settings*

Description

This function saves the current `_brand.yml` file from an existing froggeR Quarto project to your global (system-wide) froggeR configuration. This allows you to reuse brand settings across multiple projects.

Usage

```
save_brand(save_logos = TRUE)
```

Arguments

`save_logos`	Logical. Should logo files from the `logos` directory also be saved to global configuration? Default is `TRUE`.

Details

This function:

- Reads the project-level `_brand.yml` file
- Saves it to your system-wide froggeR config directory
- Optionally copies the `logos` directory for reuse in future projects
- Prompts for confirmation if a global configuration already exists

The saved configuration is stored in `rappdirs::user_config_dir('froggeR')` and will automatically be used in new froggeR projects created with `quarto_project` or `write_brand`.

Value

Invisibly returns `NULL` after saving configuration file.

See Also

`brand_settings`, `write_brand`, `save_variables`

Examples

```
# Save brand settings from current project to global config
if (interactive()) save_brand()

# Save brand settings but skip logos
if (interactive()) save_brand(save_logos = FALSE)
```

`save_variables` *Save Metadata Configuration to Global froggeR Settings*

Description

This function saves the current `_variables.yml` file from an existing froggeR Quarto project to your global (system-wide) froggeR configuration. This allows you to reuse metadata across multiple projects.

Usage

```
save_variables()
```

Details

This function:

- Reads the project-level `_variables.yml` file
- Saves it to your system-wide froggeR config directory
- Prompts for confirmation if a global configuration already exists

The saved configuration is stored in `rappdirs::user_config_dir('froggeR')` and will automatically be used in new froggeR projects created with `quarto_project` or `write_variables`.

This is useful for maintaining consistent author metadata (name, email, affiliations, etc.) across all your projects without having to re-enter it each time.

Value

Invisibly returns `NULL` after saving configuration file.

See Also

`settings`, `write_variables`, `save_brand`

Examples

```
# Save metadata from current project to global config
if (interactive()) save_variables()
```

`settings` *Manage froggeR Metadata Settings*

Description

Interactively create or update your froggeR metadata for reuse across Quarto projects.

Usage

```
settings()
```

Details

This function provides an interactive workflow to:

- Create new metadata (name, email, ORCID, affiliations, etc.)
- Update existing metadata from project or global configurations
- View current metadata
- Save to project-level, global (system-wide), or both locations

Metadata is stored as YAML and automatically used by `write_variables` in future Quarto projects.

Value

Invisibly returns `NULL`. Called for side effects.

See Also

`brand_settings`, `write_variables`, `save_variables`

Examples

```
# Only run in an interactive environment
if (interactive()) froggeR::settings()
```

`write_brand` *Write Brand YAML for Quarto Projects*

Description

This function creates a `_brand.yml` file in a Quarto project and opens it for editing.

Usage

```
write_brand(path = here::here(), restore_logos = TRUE)
```

Arguments

`path`	Character. Path to the project directory. Default is current project root via `here`.
`restore_logos`	Logical. Restore logo content from system configuration. Default is `TRUE`.

Details

The function will attempt to use the current froggeR settings from the global config path. If no global configurations exist, a template `_brand.yml` will be created.

In interactive sessions, the file is opened in the default editor for immediate customization.

Value

Invisibly returns the path to the created file.

See Also

`brand_settings`, `save_brand`, `settings`

Examples

```
# Write the _brand.yml file
if (interactive()) {
  temp_dir <- tempdir()
  # In an interactive session, this would also open the file for editing.
  write_brand(temp_dir)
  # Verify the file was created
  file.exists(file.path(temp_dir, "_brand.yml"))
}
```

`write_ignore` *Create an Enhanced .gitignore File*

Description

This function creates a `.gitignore` file with either a minimal or aggressive set of ignore rules and opens it for editing.

Usage

```
write_ignore(path = here::here(), aggressive = FALSE)
```

Arguments

`path`	Character. Path to the project directory. Default is current project root via `here`.
`aggressive`	Logical. If `TRUE`, creates a comprehensive `.gitignore` with aggressive rules for sensitive data. If `FALSE` (default), creates a minimal `.gitignore` suitable for most R projects.

Details

The `aggressive = TRUE` template includes comprehensive rules for common data file types and sensitive information.

WARNING: Always consult your organization's data security team before using git with any sensitive or protected health information (PHI). This template helps prevent accidental data exposure but should not be considered a complete security solution.

Value

Invisibly returns the path to the created file.

See Also

`quarto_project`, `write_quarto`

Examples

```
# Create a temporary directory for testing
tmp_dir <- tempdir()

# Write a minimal .gitignore file (default)
write_ignore(path = tmp_dir)

# Clean up the first file before creating the next one
unlink(file.path(tmp_dir, ".gitignore"))

# Write an aggressive .gitignore file
write_ignore(path = tmp_dir, aggressive = TRUE)

# Clean up
unlink(file.path(tmp_dir, ".gitignore"))
```

`write_notes` *Create a Dated Progress Notes File*

Description

This function streamlines project documentation by creating a dated progress notes file and opening it for editing.

Usage

```
write_notes(path = here::here())
```

Arguments

`path`	Character. Path to the project directory. Default is current project root via `here`.

Details

The `dated_progress_notes.md` file is initialized with the current date and is designed to help track project milestones chronologically. This file supports markdown formatting for rich documentation of project progress.

Value

Invisibly returns the file path after creating a dated progress notes file.

See Also

`quarto_project`, `write_readme`

Examples

```
# Create a temporary directory for testing
tmp_dir <- tempdir()

# Write the progress notes file
write_notes(path = tmp_dir)

# Confirm the file was created
file.exists(file.path(tmp_dir, "dated_progress_notes.md"))

# Clean up
unlink(file.path(tmp_dir, "dated_progress_notes.md"))
```

`write_quarto` *Create a New Quarto Document*

Description

This function creates a new Quarto document (`.qmd` file) with either a custom or standard YAML header and opens it for editing.

Usage

```
write_quarto(filename = "Untitled-1", path = here::here(), example = FALSE)
```

Arguments

`filename`	Character. The name of the file without the `.qmd` extension. Only letters, numbers, hyphens, and underscores are allowed. Default is `"Untitled-1"`.
`path`	Character. Path to the project directory. Default is current project root via `here`.
`example`	Logical. If `TRUE`, creates a Quarto document with examples and ensures that auxiliary files (`_variables.yml`, `_quarto.yml`, `custom.scss`) exist in the project. Default is `FALSE`.

Details

When `example` = `TRUE`, the function automatically creates necessary auxiliary files if they don't exist. The created document includes example content demonstrating cross-references, links, and bibliography integration.

Value

Invisibly returns the path to the created Quarto document.

See Also

`quarto_project`, `write_variables`, `write_brand`

Examples

```
if (interactive()) {
  # Create a temporary directory for testing
  tmp_dir <- tempdir()

  # Write a Quarto document with examples
  write_quarto(path = tmp_dir, filename = "analysis", example = TRUE)

  # Verify the file was created
  file.exists(file.path(tmp_dir, "analysis.qmd"))

  # Clean up
  unlink(list.files(tmp_dir, full.names = TRUE), recursive = TRUE)
}
```

`write_readme` *Create a Project README File*

Description

This function streamlines project documentation by creating a `README.md` file and opening it for editing.

Usage

```
write_readme(path = here::here())
```

Arguments

`path`	Character. Path to the project directory. Default is current project root via `here`.

Details

The `README.md` template includes structured sections for:

- Project description (study name, principal investigator, author)
- Project setup steps for reproducibility
- File and directory descriptions
- Miscellaneous project notes

Value

Invisibly returns the path to the created file.

See Also

quarto_project, write_notes

Examples

```
# Create a temporary directory for testing
tmp_dir <- tempdir()

# Write the README file
write_readme(path = tmp_dir)

# Confirm the file was created
file.exists(file.path(tmp_dir, "README.md"))

# Clean up
unlink(file.path(tmp_dir, "README.md"))
```

---

write_scss *Create a Quarto SCSS File*

Description

This function creates a `.scss` file for custom Quarto styling and opens it for editing.

Usage

```
write_scss(path = here::here())
```

Arguments

`path`	Character. Path to the project directory. Default is current project root via here.

Details

The function creates a `custom.scss` file with styling variables, mixins, and rules for customizing Quarto document appearance.

Value

Invisibly returns the path to the created file.

See Also

quarto_project, write_quarto

Examples

```
# Create a temporary directory for testing
tmp_dir <- tempdir()

# Write the SCSS file
write_scss(path = tmp_dir)

# Confirm the file was created
file.exists(file.path(tmp_dir, "custom.scss"))

# Clean up
unlink(file.path(tmp_dir, "custom.scss"))
```

`write_variables` *Write Variables YAML for Quarto Projects*

Description

This function creates a `_variables.yml` file in a Quarto project and opens it for editing.

Usage

```
write_variables(path = here::here())
```

Arguments

`path`	Character. Path to the project directory. Default is current project root via `here`.

Details

The function will attempt to use the current froggeR settings from the global config path. If no global configurations exist, a template `_variables.yml` will be created. This file stores reusable metadata (author name, email, ORCID, etc.) that can be referenced throughout Quarto documents.

Value

Invisibly returns the path to the created file.

See Also

`settings`, `save_variables`, `brand_settings`, `quarto_project`

Examples

```
# Write the _variables.yml file
if (interactive()) {
  temp_dir <- tempdir()
  # In an interactive session, this would also open the file for editing.
  write_variables(temp_dir)
  # Verify the file was created
  file.exists(file.path(temp_dir, "_variables.yml"))
}
```

Index