Package 'tera'

September 28, 2026

Title Generate Text and Documents with the Tera Templating Engine

Version 0.1.0

Description The 'tera' package uses 'extendr' to provide access to Vincent Prouillet's 'Tera' templating engine in Rust. Users mainly interact with a Tera R6 object, which serves as a template library with encapsulated methods for rendering templates with a given context. Template syntax supports additional logic, including built-in filters, tests, and functions, as well as loops, conditions, and inheritance. Documentation for Tera's templating syntax can be found at <https://keats.github.io/tera/>.

License MIT + file LICENSE

Encoding UTF-8

Config/rextendr/version 0.5.0

SystemRequirements Cargo (Rust's package manager), rustc

Depends R (>= 4.2)

Imports cli, R6, rlang (>= 1.1.0), yyjsonr

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

URL https://github.com/kbvernon/tera-r,
https://kbvernon.github.io/tera-r/

VignetteBuilder quarto, knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation yes

Author Kenneth Blake Vernon [aut, cre, cph] (ORCID: <https://orcid.org/0000-0003-0098-5092>)

Maintainer Kenneth Blake Vernon <kenneth.b.vernon@gmail.com>

Repository CRAN

Date/Publication 2026-09-28 14:10:02 UTC

Contents

`render-one-off` *One-Off Template Rendering*

Description

For rendering a single template file or string, it may be preferable to use these one-off rendering options.

Usage

```
render_template(path, ..., outfile = NULL, autoescape = TRUE)

render_string(string, ..., outfile = NULL, autoescape = TRUE)
```

Arguments

`path`	character scalar, path to a template file
`...`	specify context as key-value pairs where key is the template variable and value is the data to inject.
`outfile`	character scalar, the path to file where template is to be rendered. If `NULL` (the default), it will render the template file to a string in the current R session.
`autoescape`	boolean scalar, whether to autoescape HTML (default is `TRUE`).
`string`	character scalar, the template string to render.

Details

Requires a path to a template file, not a template string.

Value

'outfile" (invisibly)

Examples

```
tmp <- system.file("templates/hello.html", package = "tera")

# render to string
render_template(
  tmp,
  x = "world",
  y = "tera"
```

```
)

render_string(
  "<p>Hello {{ x }}. This is {{ y }}.</p>",
  x = "world",
  y = "tera"
)

# render to file
outfile <- file.path(tempdir(), "hello-rendered.html")
render_template(
  tmp,
  x = "world",
  y = "tera",
  outfile = outfile
)

readLines(outfile, warn = FALSE)
```

`Tera` *R6 class for Tera templating engine*

Description

A Tera templating engine for rendering templates with contexts consisting of variables and values. Similar to glue, but with additional template logic.

The glob pattern `templates/*.html` will match all files with the .html extension located directly inside the `templates` folder, while `templates/**/*.html` will match all files with the .html extension directly inside or in a subdirectory of `templates`. The default naming convention is to give each template their full relative path from `templates` (or whatever the template directory is called).

When initialized with a glob, template names take the path of the file template relative to the base of the template directory. For example, if you pass `templates/**/*.html`, the template file `templates/blog/post.html` would have the name `blog/post.html`. When specified manually, all templates must be named. The same is true of context variables.

It is possible to add variables to a global context that can be accessed in any template. See the `set_globals()` method.

Methods

Public methods:

- `Tera$new()`
- `Tera$print()`
- `Tera$add_file_templates()`
- `Tera$add_string_templates()`
- `Tera$reload()`
- `Tera$render_template()`

- Tera$render_string()
- Tera$render_component()
- Tera$templates()
- Tera$components()
- Tera$variables()
- Tera$autoescape_on()
- Tera$autoescape_off()
- Tera$set_globals()
- Tera$globals()
- Tera$clear_globals()
- Tera$set_delimiters()
- Tera$delimiters()

`Tera$new()`: Initialize a Tera template engine

Usage:

```
Tera$new(glob = NULL)
```

Arguments:

`glob` character scalar, a glob pattern with `*` wildcards indicating a potentially nested directory containing multiple file templates. If `NULL` (the default), a `Tera` engine with an empty library is initialized.

Returns: an R6 `Tera` template engine.

`Tera$print()`: Print Tera

Usage:

```
Tera$print(n = 10L, ...)
```

Arguments:

`n` integer scalar, number of templates to print (default is 10L)

`...` ignored

Returns: Self (invisibly)

`Tera$add_file_templates()`: Add file templates to library from file paths.

Usage:

```
Tera$add_file_templates(...)
```

Arguments:

`...` specify list of templates as `key = value` pairs where key is the name of the template and value is the path to the template on file.

Returns: Self (invisibly)

`Tera$add_string_templates()`: Add templates to library from character strings.

Usage:

```
Tera$add_string_templates(...)
```

Arguments:

`...` specify list of templates as `key = value` pairs where key is the name of the template and value is a string template.

Returns: Self (invisibly)

`Tera$reload()`: Re-parse all file templates found in the glob given to Tera, including any new file templates that were added. Templates added manually are preserved.

Usage:

```
Tera$reload()
```

Returns: Self (invisibly)

`Tera$render_template()`: Render specified template file to string or file if `outfile` is specified.

Usage:

```
Tera$render_template(template, ..., block = NULL, outfile = NULL)
```

Arguments:

`template` character scalar, the name of the template to render.

`...` specify context as `key = value` pairs where key is the template variable and value is the data to inject.

`block` character scalar, if not `NULL`, interpolation will be restricted to the specified block in template (default is `NULL`).

`outfile` character scalar, the path to file where template is to be rendered (default is `NULL`).

Returns: if `outfile = NULL` (default), the rendered string; otherwise, `outfile` (invisibly).

`Tera$render_string()`: Render specified template string to string or file if `outfile` is specified.

Usage:

```
Tera$render_string(string, ..., outfile = NULL, autoescape = TRUE)
```

Arguments:

`string` character scalar, the template string to render.

`...` specify context as `key = value` pairs where key is the template variable and value is the data to inject.

`outfile` character scalar, the path to file where template is to be rendered (default is `NULL`).

`autoescape` boolean scalar, whether to autoescape HTML (default is `TRUE`).

Returns: if `outfile = NULL` (default), the rendered string; otherwise, `outfile` (invisibly).

`Tera$render_component()`: Render specified component to string or file if `outfile` is specified.

Usage:

```
Tera$render_component(
  component,
  ...,
  body = NULL,
  outfile = NULL,
  autoescape = TRUE
)
```

Arguments:

`component` character scalar, the name of the component to render.

`...` specify context as `key = value` pairs where key is the template variable and value is the data to inject.

`body` character scalar, optional, passed to component `{{ body }}` variable.

`outfile` character scalar, the path to file where template is to be rendered (default is `NULL`).

`autoescape` boolean scalar, whether to autoescape HTML (default is `TRUE`).

Returns: if `outfile = NULL` (default), the rendered string; otherwise, `outfile` (invisibly).

`Tera$templates()`**:** List current templates in library.

Usage:

```
Tera$templates()
```

Returns: a sorted `character` vector of template names

`Tera$components()`**:** List current components in library.

Usage:

```
Tera$components()
```

Returns: a sorted `character` vector of component names

`Tera$variables()`**:** List all variables in template.

Usage:

```
Tera$variables(template)
```

Arguments:

`template` character scalar, the name of the template.

Returns: a sorted `character` vector of variable names

`Tera$autoescape_on()`**:** Turn on autoescaping of HTML. This only applies to templates whose names end with ".html", ".htm", or ".xml". Autoescaping is on by default.

Usage:

```
Tera$autoescape_on()
```

Returns: Self (invisibly)

`Tera$autoescape_off()`**:** Turn off autoescaping of HTML. This only applies to templates whose names end with ".html", ".htm", or ".xml". Autoescaping is on by default.

Usage:

```
Tera$autoescape_off()
```

Returns: Self (invisibly)

`Tera$set_globals()`**:** Add a set of variables to a global context that can then be accessed by every template. If a variable with the same name already exists, it is overridden.

Usage:

```
Tera$set_globals(...)
```

Arguments:

`...` specify context as `key = value` pairs where key is the template variable and value is the data to inject.

Returns: Self (invisibly)

`Tera$globals()`: List current variables in global context.

Usage:

```
Tera$globals()
```

Returns: a sorted `character` vector of variable names

`Tera$clear_globals()`: Remove all variables from global context. Resets to an empty context.

Usage:

```
Tera$clear_globals()
```

Returns: Self (invisibly)

`Tera$set_delimiters()`: Set the delimiters used to mark up templates, which is useful for templating files that themselves contain `{{`, such as LaTeX. Each delimiter must be exactly two bytes long, the three start delimiters must differ from each other, and any delimiter left `NULL` is unchanged.

Delimiters must be set before any templates are added, so this can only be called on an engine with an empty library.

Usage:

```
Tera$set_delimiters(
  block_start = NULL,
  block_end = NULL,
  variable_start = NULL,
  variable_end = NULL,
  comment_start = NULL,
  comment_end = NULL
)
```

Arguments:

`variable_start`, `variable_end` character scalar, delimiters for variables (defaults are `"{{"` and `"}}"`).

`comment_start`, `comment_end` character scalar, delimiters for comments (defaults are `"{#"` and `"#}"`).

Returns: Self (invisibly)

`Tera$delimiters()`: List the delimiters currently used to mark up templates.

Usage:

```
Tera$delimiters()
```

Returns: a named `list` of delimiters

Examples

```
# - setup -----
# initialize empty Tera engine
tera <- Tera$new()
tera

# initialize Tera engine from directory with glob
template_dir <- system.file("templates", package = "tera")
glob <- "**/*.html"
tera <- Tera$new(file.path(template_dir, glob))
tera

# add templates manually from file
tera <- Tera$new()
tera$add_file_templates(
  "base.html" = file.path(template_dir, "base.html"),
  "index.html" = file.path(template_dir, "index.html")
)
tera

# add templates manually from string
tera$add_string_templates(
  img = '<img src="{{ img_src }}">'
)
tera

# reload templates from glob
example_dir <- file.path(tempdir(), "tera-templates")
dir.create(example_dir)

file.copy(
  from = file.path(template_dir, "base.html"),
  to = file.path(example_dir, "base.html")
)

tera <- Tera$new(file.path(example_dir, "*.html"))
tera

file.copy(
  from = file.path(template_dir, "index.html"),
  to = file.path(example_dir, "index.html")
)

tera$reload()

# - rendering -----
tera <- Tera$new(file.path(template_dir, glob))

cat(
  readLines(file.path(template_dir, "index.html")),
  sep = "\n"
)
```

```
# render a template file
tera$render_template(
  "index.html",
  title = "Index",
  p = "Welcome to my awesome homepage.",
  owner = "Blake"
)

# render a template block
tera$render_template(
  "index.html",
  title = "Index",
  p = "Welcome to my awesome homepage.",
  owner = "Bob",
  block = "content"
)

# render a template string
tera$render_string(
  '<img src="{{ img_src }}">',
  img_src = "foo/bar.svg"
)

# render a component
cat(
  readLines(file.path(template_dir, "components.html")),
  sep = "\n"
)

tera$render_component(
  "button",
  label = "Primary Button",
  variant = "primary"
)

# render a component with body
tera$render_component(
  "widget",
  title = "foo",
  body = "<p>This is a widget!</p>"
)

# - inspection -----
# list all templates in library
tera$templates()

# list all variables in template
tera$variables("index.html")

# list all components in library
tera$components()
```

```
# - global context -----
# add variables to global context
tera$set_globals(
  base_url = "https://www.example.com",
  site_title = "My Website",
  description = "This is my awesome website!"
)

# list all variables in global context
tera$globals()

# clear global context
tera$clear_globals()
tera$globals()

# - autoescape -----
tera <- Tera$new()

template_dir <- system.file("templates", package = "tera")
tera$add_file_templates(
  "hello" = file.path(template_dir, "hello.html"),
  "hello.html" = file.path(template_dir, "hello.html")
)

# not recognized as html
tera$render_template(
  "hello",
  x = "&world",
  y = "an apostrophe, '"
)

# html
tera$render_template(
  "hello.html",
  x = "&world",
  y = "an apostrophe, '"
)

# turn off autoescape
tera$autoescape_off()

tera$render_template(
  "hello.html",
  x = "&world",
  y = "an apostrophe, '"
)

# - delimiters -----
# must be set before any templates are added
tera <- Tera$new()

tera$set_delimiters(
  variable_start = "<<",
```

```
  variable_end = ">>"
)

tera$delimiters()

tera$render_string("<< greeting >>, world!", greeting = "Hello")
```

Index