Package 'roxygen2'

January 22, 2024

Title In-Line Documentation for R

Version 7.3.1

Description Generate your Rd documentation, 'NAMESPACE' file, and collation field using specially formatted comments. Writing documentation in-line with code makes it easier to keep your documentation up-to-date as your requirements change. 'roxygen2' is inspired by the 'Doxygen' system for C++.

License MIT + file LICENSE

URL https://roxygen2.r-lib.org/, https://github.com/r-lib/roxygen2

BugReports https://github.com/r-lib/roxygen2/issues

Depends R (>= 3.6)

Imports brew, cli (>= 3.3.0), commonmark, desc (>= 1.2.0), knitr, methods, pkgload (>= 1.0.2), purrr (>= 1.0.0), R6 (>= 2.1.2), rlang (>= 1.0.6), stringi, stringr (>= 1.0.0), utils, withr, xml2

Suggests covr, R.methodsS3, R.oo, rmarkdown (>= 2.16), testthat (>= 3.1.2), yaml

LinkingTo cpp11

VignetteBuilder knitr

Config/Needs/development testthat

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Config/testthat/parallel TRUE

Encoding UTF-8

Language en-GB

RoxygenNote 7.3.0.9000

NeedsCompilation yes

Author Hadley Wickham [aut, cre, cph]
(<https://orcid.org/0000-0003-4757-117X>),
Peter Danenberg [aut, cph],
Gábor Csárdi [aut],
Manuel Eugster [aut, cph],
Posit Software, PBC [cph, fnd]

Maintainer Hadley Wickham <hadley@posit.co>

Repository CRAN

Date/Publication 2024-01-22 21:10:03 UTC

R topics documented:

`load` *Load package code*

Description

roxygen2 is a dynamic documentation system, which means it works with the objects inside your package, not just the source code used to create them. These functions offer various ways of loading your package to suit various constraints:

- `load_pkgload()` uses `pkgload::load_all()` to simulate package loading as closely as we know how. It offers high fidelity handling of code that uses S4, but requires that the package be compiled.
- `load_source()` simulates package loading by attaching packages listed in `Depends` and `Imports`, then sources all files in the `R/` directory. This was the default strategy used in roxygen2 6.0.0 and earlier; it's primary advantage is that it does not need compilation.
- `load_installed()` uses the installed version of the package. Use this strategy if you have installed a development version of the package already. This is the highest fidelity strategy, but requires work outside of roxygen2.

You can change the default strategy for your function with roxygen2 `load` option. Override the default off `pkgload` to use the `source` or `installed` strategies:

```
Roxygen: list(load = "source")
```

Usage

```
load_pkgload(path)

load_installed(path)

load_source(path)
```

Arguments

`path`	Path to source package

`namespace_roclet` *Roclet: make* `NAMESPACE`

Description

This roclet automates the production of a `NAMESPACE` file, which controls the functions imported and exported by your package, as described in Writing R extensions.

The `NAMESPACE` is generated in two passes: the first generates only import directives (because this can be computed without evaluating package code), and the second generates everything (after the package has been loaded).

See `vignette("namespace")` for details.

Usage

```
namespace_roclet()
```

See Also

tags-namespace for tags that generate `NAMESPACE` directives.

Examples

```
# The most common namespace tag is @export, which declares that a function
# is part of the external interface of your package
#' @export
foofy <- function(x, y, z) {
}

# You'll also often find global imports living in a file called
# R/{package}-package.R.
#' @importFrom magrittr %>%
#' @import rlang
NULL
```

rd_roclet *Roclet: make Rd files*

Description

This roclet is the workhorse of roxygen2, producing the `.Rd` files that R uses to document functions, datasets, packages, classes, and more. See `vignette("rd")` for details.

Generally you will not call this function directly but will instead use `roxygenise()` specifying the rd roclet.

Usage

```
rd_roclet()
```

See Also

tags-rd, tags-rd-other, tags-reuse, tags-index-crossref for tags provided by this roclet.

Examples

```
#' The length of a string (in characters)
#'
#' @param x A character vector.
#' @returns An integer vector the same length as `x`.
#'   `NA` strings have `NA` length.
#' @seealso [nchar()]
#' @export
#' @examples
#' str_length(letters)
#' str_length(c("i", "like", "programming", NA))
str_length <- function(x) {
}
```

roxygenize *Process a package with the Rd, namespace and collate roclets*

Description

This is the workhorse function that uses roclets, the built-in document transformation functions, to build all documentation for a package. See the documentation for the individual roclets, `rd_roclet()`, `namespace_roclet()`, and for `update_collate()`, for more details.

Usage

```
roxygenize(package.dir = ".", roclets = NULL, load_code = NULL, clean = FALSE)

roxygenise(package.dir = ".", roclets = NULL, load_code = NULL, clean = FALSE)
```

Arguments

package.dir	Location of package top level directory. Default is working directory.
roclets	Character vector of roclet names to use with package. The default, NULL, uses the roxygen roclets option, which defaults to c("collate", "namespace", "rd").
load_code	A function used to load all the R code in the package directory. The default, NULL, uses the strategy defined by the load roxygen option, which defaults to load_pkgload(). See load for more details.
clean	If TRUE, roxygen will delete all files previously created by roxygen before running each roclet.

Details

Note that roxygen2 is a dynamic documentation system: it works by inspecting loaded objects in the package. This means that you must be able to load the package in order to document it: see load for details.

Value

NULL

---

tags-index-crossref *Tags for indexing and cross-references*

Description

Learn the full details in vignette('index-crossref').

Key tags:

- @aliases ${1:alias}: Add additional aliases to the topic. Use NULL to suppress the default alias automatically generated by roxygen2.
- @concept ${1:concept}: Add additional keywords or phrases to be included in the help.search() index. Each @concept should be a single term or phrase.
- @family ${1:family name}: Generate @seealso entries to all other functions in family name.
- @keywords ${1:keyword}: Add a standardised keyword, indexed by help.search(). These are generally not useful apart from @keywords internal which flags the topic as internal and removes from topic indexes.
- @references ${1:reference}: Pointers to the related literature. Usually formatted like a bibliography.
- @seealso [${1:func}()]: Link to other related functions or urls. Usually a sentence or two, or a bulleted list.

Other less frequently used tags:

- @backref ${1:path}: Manually override the backreference that points from the .Rd file back to the source .R file. Only needed when generating code.

Usage

```
#' @aliases ${1:alias}
#' @backref ${1:path}
#' @concept ${1:concept}
#' @family ${1:family name}
#' @keywords ${1:keyword}
#' @references ${1:reference}
#' @seealso [${1:func}()]
```

See Also

Other documentation tags: tags-rd, tags-rd-other, tags-reuse

---

tags-namespace *Tags for managing the* NAMESPACE

Description

Learn the full details in `vignette('namespace')`.

Key tags:

- `@export`: Export this function, method, generic, or class so it's available outside of the package.
- `@exportS3Method ${1:package}::${2:generic}`: Export an S3 method. Only needed when the method is for a generic from a suggested package.
- `@importFrom ${1:package} ${2:function}`: Import specific functions from a package.
- `@useDynLib ${1:package}`: Import compiled code from another package.

Other less frequently used tags:

- `@evalNamespace ${1:r-code}`: Evaluate arbitrary code in the package namespace and insert the results into the NAMESPACE. Should return a character vector of directives.
- `@exportClass ${1:class}`: Export an S4 class. For expert use only; in most cases you should use `@export` so roxygen2 can automatically generate the correct directive.
- `@exportMethod ${1:generic}`: Export S4 methods. For expert use only; in most cases you should use `@export` so roxygen2 can automatically generate the correct directive.
- `@exportPattern ${1:pattern}`: Export all objects matching a regular expression.
- `@import ${1:package}`: Import all functions from a package. Use with extreme care.
- `@importClassesFrom ${1:package} ${2:class}`: Import S4 classes from another package.
- `@importMethodsFrom ${1:package} ${2:generic}`: Import S4 methods from a package.
- `@rawNamespace ${1:namespace directives}`: Insert literal text directly into the NAMESPACE.

Usage

```
#' @evalNamespace ${1:r-code}
#' @export
#' @exportClass ${1:class}
#' @exportMethod ${1:generic}
#' @exportPattern ${1:pattern}
#' @exportS3Method ${1:package}::${2:generic}
#' @import ${1:package}
#' @importClassesFrom ${1:package} ${2:class}
#' @importFrom ${1:package} ${2:function}
#' @importMethodsFrom ${1:package} ${2:generic}
#' @rawNamespace ${1:namespace directives}
#' @useDynLib ${1:package}
```

`tags-rd` *Tags for documenting functions*

Description

Learn the full details in `vignette('rd')`.

Key tags:

- `@description${1:A short description...}` : A short description of the purpose of the function. Usually around a paragraph, but can be longer if needed.
- `@example ${1:path}.R`: Embed examples stored in another file.
- `@examples${1:# example code}` : Executable R code that demonstrates how the function works. Code must run without error.
- `@examplesIf ${1:condition}${2:# example code}` : Run examples only when `condition` is `TRUE`.
- `@noRd`: Suppress `.Rd` generation for a block. Use for documentation blocks that should only be visible in the source code.
- `@param ${1:name} ${2:description}`: Describe a function input. Should describe acceptable input types and how it affects the output. `description` is usually one or two sentences but can be as long as needed. Document multiple arguments by separating their names with commas without spaces.
- `@returns ${1:description}`: Describe the function's output. Typically will be a 1-2 sentence description of the output type, but might also include discussion of important errors or warnings.
- `@title ${1:title}`: A one-line description of the function shown in various indexes. An explicit `@title` is not usually needed as by default it is taken from the first paragraph in the roxygen block.
- `@usage ${1:fun}(${2:arg1, arg2 = default, ...})`: Override the default usage generated by roxygen2. Only needed when roxygen2 fails to correctly derive the usage of your function.

Other less frequently used tags:

- `@details${1:Additional details...}` : Additional details about the function. Generally superseded by instead using a level 1 heading.
- `@rawRd ${1:rd}`: Insert literal text directly into the `.Rd` file.
- `@return ${1:description}`: Describe the function's output. Superseded in favour of `@returns`.

Usage

```
#' @description${1:A short description...}

#' @details${1:Additional details...}

#' @example ${1:path}.R
#' @examples${1:# example code}

#' @examplesIf ${1:condition}${2:# example code}

#' @noRd
#' @param ${1:name} ${2:description}
#' @rawRd ${1:rd}
#' @return ${1:description}
#' @returns ${1:description}
#' @title ${1:title}
#' @usage ${1:fun}(${2:arg1, arg2 = default, ...})
```

See Also

Other documentation tags: tags-index-crossref, tags-rd-other, tags-reuse

---

`tags-rd-formatting` *Tags related to markdown support*

Description

Learn the full details in `vignette('rd-formatting')`.

Other less frequently used tags:

- `@md`: Force markdown processing for a block.
- `@noMd`: Suppress markdown processing for a block.
- `@section ${1:section title}:` : Add an arbitrary section to the documentation. Now generally superseded in favour of using a level 1 heading.

Usage

```
#' @md
#' @noMd
#' @section ${1:section title}:
```

tags-rd-other *Tags for documenting datasets and classes*

Description

Learn the full details in `vignette('rd-other')`.

Key tags:

- `@field ${1:name} ${2:description}`: Describe a R6 or refClass field.
- `@format ${1:description}`: Describe the type/shape of a dataset. If the dataset is a data frame, include a description of each column. If not supplied, will be automatically generated by `object_format()`.
- `@method ${1:generic} ${2:class}`: Force a function to be recognised as an S3 method. This affects the default usage and the `NAMESPACE` directive produced by `@export`. Only needed if automatic detection fails.
- `@slot ${1:name} ${2:description}`: Describe the slot of an S4 class.
- `@source ${1:description}`: Describe where the dataset came from. Provide a link to the original source (if possible) and briefly describe any manipulation that you performed when importing the data.

Usage

```
#' @field ${1:name} ${2:description}
#' @format ${1:description}
#' @method ${1:generic} ${2:class}
#' @slot ${1:name} ${2:description}
#' @source ${1:description}
```

See Also

Other documentation tags: `tags-index-crossref`, `tags-rd`, `tags-reuse`

tags-reuse *Tags that help you reuse documentation*

Description

Learn the full details in `vignette('reuse')`.

Key tags:

- `@describeIn ${1:destination} ${2:description}`: Document a function or method in the `destination` topic.

- `@inherit ${1:source} ${2:components}`: Inherit one or more documentation components from another topic. If `components` is omitted, all supported components will be inherited. Otherwise, specify individual components to inherit by picking one or more of `params`, `return`, `title`, `description`, `details`, `seealso`, `sections`, `references`, `examples`, `author`, `source`, `note`, and `format`.
- `@inheritDotParams ${1:source} ${2:arg1 arg2 arg3}`: Automatically generate documentation for `...` when you're passing dots along to another function.
- `@inheritParams ${1:source}`: Inherit argument documentation from another function. Only inherits documentation for arguments that aren't already documented locally.
- `@inheritSection ${1:source} ${2:section name}`: Inherit a specific named section from another topic.
- `@order ${1:number}`: Override the default (lexigraphic) order in which multiple blocks are combined into a single topic.
- `@rdname ${1:topic-name}`: Override the file name of generated `.Rd` file. Can be used to combine multiple blocks into a single documentation topic.

Other less frequently used tags:

- `@eval ${1:r-code}`: Evaluate arbitrary code in the package namespace and insert the results back into the block. Should return a character vector of lines.
- `@evalRd ${1:r-code}`: Evaluate arbitrary code in the package namespace and insert the results back as into the block. Should return a character vector of lines.
- `@includeRmd man/rmd/${1:filename}.Rmd`: Insert the contents of an `.Rmd` into the current block. Superseded in favour of using a code chunk with a child document.
- `@template ${1:path-to-template}`: Use a roxygen2 template. Now superseded in favour of inline R code.
- `@templateVar ${1:name} ${2:value}`: Define variables for use in a roxygen2 template.

Usage

```
#' @describeIn ${1:destination} ${2:description}
#' @eval ${1:r-code}
#' @evalRd ${1:r-code}
#' @includeRmd man/rmd/${1:filename}.Rmd
#' @inherit ${1:source} ${2:components}
#' @inheritDotParams ${1:source} ${2:arg1 arg2 arg3}
#' @inheritParams ${1:source}
#' @inheritSection ${1:source} ${2:section name}
#' @order ${1:number}
#' @rdname ${1:topic-name}
#' @template ${1:path-to-template}
#' @templateVar ${1:name} ${2:value}
```

See Also

Other documentation tags: tags-index-crossref, tags-rd, tags-rd-other

`update_collate` *Update Collate field in DESCRIPTION*

Description

By default, R loads files in alphabetical order. Unfortunately not every alphabet puts letters in the same order, so you can't rely on alphabetic ordering if you need one file loaded before another. (This usually doesn't matter but is important for S4, where you need to make sure that classes are loaded before subclasses and generics are defined before methods.). You can override the default alphabetical ordering with `@include before.R`, which specify that `before.R` must be loaded before the current file.

Generally, you will not need to run this function yourself; it should be run automatically by any package that needs to load your R files in collation order.

Usage

```
update_collate(base_path)
```

Arguments

`base_path`	Path to package directory.

Collate

This is not a roclet because roclets need the values of objects in a package, and those values can not be generated unless you've sourced the files, and you can't source the files unless you know the correct order.

If there are no `@include` tags, roxygen2 will leave collate as is. This makes it easier to use roxygen2 with an existing collate directive, but if you remove all your `@include` tags, you'll need to also manually delete the collate field.

Examples

```
#' If `example-a.R', `example-b.R' and `example-c.R' live in R/
#' and we're in `example-a.R`, then the following @include statement
#' ensures that example-b and example-c are sourced before example-a.
#' @include example-b.R example-c.R
NULL
```

Index