Package 'roxygen2'

May 13, 2022

Title In-Line Documentation for R

Version 7.2.0

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 (>= 3.3)

Imports brew, cli (>= 3.3.0), commonmark, desc (>= 1.2.0), digest, knitr, methods, pkgload (>= 1.0.2), purrr (>= 0.3.3), R6 (>= 2.1.2), rlang (>= 1.0.0), stringi, stringr (>= 1.0.0), utils, withr, xml2

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

LinkingTo cpp11

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

RoxygenNote 7.1.2.9000

SystemRequirements C++11

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], RStudio [cph, fnd]

Maintainer Hadley Wickham <hadley@rstudio.com>

Repository CRAN

Date/Publication 2022-05-13 07:30:02 UTC

R topics documented:

`is_s3_generic` *Determine if a function is an S3 generic or S3 method.*

Description

`is_s3_generic` compares name to `.knownS3Generics` and `.S3PrimitiveGenerics`, then looks at the function body to see if it calls `UseMethod()`.

`is_s3_method` builds names of all possible generics for that function and then checks if any of them actually is a generic.

Usage

```
is_s3_generic(name, env = parent.frame())

is_s3_method(name, env = parent.frame())
```

Arguments

`name`	Name of function.
`env`	Base environment in which to look for function definition.

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

Other roclets: `rd_roclet()`, `vignette_roclet()`

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
```

---

object_format *Default format for data*

Description

This function is called to generate the default "Format" section for each data object. The default implementation will return the class and dimension information.

Usage

```
object_format(x)
```

Arguments

x	A data object

Value

A `character` value with valid `Rd` syntax, or `NULL`.

rd_roclet *Roclet: make Rd files.*

Description

This roclet is the workhorse of roxygen, producing the .Rd files that R uses to document that functions, datasets, packages, classes, and other objects.

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

Other roclets: namespace_roclet(), vignette_roclet()

Examples

```
#' The length of a string (in characters)
#'
#' @param x String input character vector
#' @return 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) {
}
```

roclet_find *Create a roclet from a string.*

Description

This provides a flexible way of specifying a roclet in a string.

Usage

```
roclet_find(x)
```

Arguments

x	Arbitrary R code evaluated in roxygen2 package.

Examples

```
# rd, namespace, and vignette work for backward compatibility
roclet_find("rd")

# But generally you should specify the name of a function that
# returns a roclet
roclet_find("rd_roclet")

# If it lives in another package, you'll need to use ::
roclet_find("roxygen2::rd_roclet")

# If it takes parameters (which no roclet does currently), you'll need
# to call the function
roclet_find("roxygen2::rd_roclet()")
```

---

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

---

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
```

`vignette_roclet` *Re-build outdated vignettes.*

Description

This rebuilds outdated vignettes with tools::buildVignette. By default, it will rebuild all vignettes if the source file is newer than the output pdf or html. (This means it will automatically re-build the vignette if you change the vignette source, but *not* when you change the R code). If you want finer control, add a Makefile to `vignettes/` and roxygen2 will use that instead.

Usage

```
vignette_roclet()
```

Details

To prevent RStudio from re-building the vignettes again when checking your package, add `--no-build-vignettes` to the "Build Source Package" field in your project options.

See Also

Other roclets: `namespace_roclet()`, `rd_roclet()`

Index