

# Package ‘document’

July 22, 2025

**Title** Run 'roxygen2' on (Chunks of) Single Code Files

**Version** 4.0.1

**Description** Have you ever been tempted to create 'roxygen2'-style documentation comments for one of your functions that was not part of one of your packages (yet)?  
This is exactly what this package is about: running 'roxygen2' on (chunks of) a single code file.

**Depends** R (>= 3.3.0)

**License** BSD\_2\_clause + file LICENSE

**URL** <https://gitlab.com/fvafrcu/document>

**Encoding** UTF-8

**Imports** callr, checkmate, desc, fritools, rcmdcheck, roxygen2, rstudioapi

**Suggests** knitr, pkgload, rmarkdown, RUnit, testthat

**VignetteBuilder** knitr

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Andreas Dominik Cullmann [aut, cre]

**Maintainer** Andreas Dominik Cullmann <fvafrcu@mailbox.org>

**Repository** CRAN

**Date/Publication** 2025-05-24 23:40:02 UTC

## Contents

document-package . . . . .	2
document . . . . .	3
man . . . . .	5
usage . . . . .	6
<b>Index</b>	<b>7</b>

## Description

Have you ever been tempted to create **roxygen2**-style documentation comments for one of your functions that was not part of one of your packages (yet)? This is exactly what this package is about: running `roxygen2::roxygenize` on (chunks of) a single code file.

This package enables you to

1. create function documentation with **roxygen2**
2. detect code/documentation mismatches
3. save the documentation to disk
4. view the documentation in your interactive R session

You will probably be looking for `document` and `man`, the remaining functions are mainly for internal use.

## Details

R is a programming language that supports and checks documentation for program libraries (called ‘packages’). The package **roxygen2** provides a tool for creating documentation from annotated source code - much like `doxygen`, `javadoc` and `docstrings/pydoc` do.

And R is a free software environment for statistical computing and graphics, used by people like me who start out hacking down code, eventually pouring chunks of code into functions (and sometimes even ending up creating and documenting packages). Along that work flow you cannot use R’s documentation system, let alone **roxygen2**, unless you have come to forge your code into a package.

I am fully aware of the fact that **roxygen2** is meant to document packages, not single code chunks. So should you. Nevertheless I feel the temptation to use **roxygen2**-style comments in code chunks that are not part of any package. And to convert them to pdf for better readability.

## Warning

This package writes to disk, so **never** run as superuser.

## Note

This package is basically a wrapper to

1. **roxygen2**. It internally creates a temporary package from the code file provided (using `utils::package.skeleton`) which it then passes to `roxygen2::roxygenize`.
2. R CMD commands run by `callr`.

## Author(s)

**Maintainer:** Andreas Dominik Cullmann <fvafrcu@mailbox.org>

**See Also**

**docstring** (<https://cran.r-project.org/package=docstring>) also creates temporary help pages as well but using a different technical approach (allowing you to view them in the RStudio help pane). But it creates them from python style docstring-like comments it then parses into **roxy-gen2**. And it does not write to file so far.

document

*Document (Chunks of) an R Code File***Description**

Document (Chunks of) an R Code File

**Usage**

```
document(
  file_name,
  working_directory = NULL,
  output_directory = tempdir(),
  dependencies = NULL,
  sanitize_Rd = TRUE,
  runit = FALSE,
  check_package = TRUE,
  check_as_cran = check_package,
  stop_on_check_not_passing = check_package,
  clean = FALSE,
  debug = TRUE,
  ...
)
```

**Arguments**

<code>file_name</code>	The name of the R code file to be documented.
<code>working_directory</code>	A working directory. Keep the default.
<code>output_directory</code>	The directory to put the documentation into. You might want to use <code>dirname(file_name)</code> .
<code>dependencies</code>	A character vector of package names the functions depend on.
<code>sanitize_Rd</code>	Remove strange characters from Rdconv?
<code>runit</code>	Convert the text received from the help files if running <b>RUnit</b> ? Do not bother, this is for Unit testing only.
<code>check_package</code>	Run R CMD check the sources? See <b>Note</b> below.
<code>check_as_cran</code>	Use the <code>--as-cran</code> flag with R CMD check?
<code>stop_on_check_not_passing</code>	Stop if R CMD check does not pass?

<code>clean</code>	Delete the working directory?
<code>debug</code>	For internal use only: Summarize errors for travis?
<code>...</code>	Arguments passed to <code>get_lines_between_tags</code> .

### Value

A list containing

**pdf\_path** The path to the pdf file produced,

**txt\_path** The path to the text file produced,

**html\_path** The path to the html file produced,

**check\_result** The return value of `rcmdcheck::rcmdcheck()`

### Note

One of the main features of R CMD check is checking for code/documentation mismatches (it behaves pretty much like doxygen). No build system can check whether your documentation is useful, but R CMD check checks if it is formally matching your code. This check is the basic idea behind **document**. The possibility to disable the R CMD check is there to disable CPU consuming checks while testing the package. Stick with the default! And do not forget to export your functions using the line

```
#' @export
should you provide examples.
```

### Examples

```
res <- document(file_name = system.file("files", "minimal.R",
                                         package = "document"),
               check_package = FALSE) # this is for the sake of CRAN cpu
                                     # time only. _Always_ stick with the default!

# View R CMD check results. If we had set check_package to TRUE in the above
# example, we now could retrieve the check results via:
cat(res[["check_result"]][["output"]][["stdout"]], sep = "\n")
cat(res[["check_result"]][["output"]][["stderr"]], sep = "\n")

# Copy documentation to current working directory.
# This writes to your disk, so it's disabled.
# Remove or comment out the next line to enable.
if (FALSE)
  file.copy(res[["pdf_path"]], getwd())
```

---

man*Display a Help Page From a File's Documentation*

---

## Description

Display a [help](#)-like page from an existing R documentation (\*.Rd) file, a topic from a temporary package with options("document\_package\_directory") set or a topic from an R code file containing **roxygen2** documentation.

## Usage

```
man(x, topic = NA, force_Rd = FALSE)
```

## Arguments

x	One of the following: <ul style="list-style-type: none"><li>• A path to an R documentation (*.Rd) file.</li><li>• A path to a code file containing comments for <b>roxygen2</b>.</li><li>• A <a href="#">help</a> topic if options("document_package_directory") is set (by <a href="#">document</a>).</li></ul>
topic	A <a href="#">help</a> topic if x is a path to a code file containing comments for <b>roxygen2</b> .
force_Rd	if x is a file's path, then <a href="#">is_Rd_file</a> is used to decide whether the file is an R documentation file and call <a href="#">document</a> otherwise. Set to TRUE to disable this check and force the file to be assumed to be an R documentation file. <a href="#">is_Rd_file</a> may produce false negatives.

## Value

Invisibly the status of [display\\_Rd](#).

## Examples

```
document::document(file_name = system.file("files", "minimal.R",
                                             package = "document"), check_package = FALSE)
document::man("foo")
# this equivalent to
path <- system.file("files", "minimal.R", package = "document")
document::man(x = path, topic = "foo")
```

---

`usage`*Return the Usage of a Function From Within the Function*

---

**Description**

Get a usage template for a function from within the function. If you encounter misguided usage, you can display the template.

**Usage**

```
usage(n = -1, usage = FALSE)
```

**Arguments**

<code>n</code>	A negative integer giving the number of from to frames/environments to go back (passed as which to <code>sys.call</code> ). Set to -2 if you want to encapsulate the call to usage into a function (like <code>print</code> or <code>assign</code> ) within the function you want to obtain the usage for. Use the <code>&lt;-</code> assignment operator with the default, see <b>examples</b> below.
<code>usage</code>	Give this functions usage (as a usage example ...) and exit?

**Value**

A character string giving the Usage as `help` would do.

**Examples**

```
# usage with assignment operator:
foo <- function(x) {
  u <- usage()
  message("Usage is: ", u)
}
foo()

# usage without assignment operator:
bar <- function(x) {
  message(usage(n = -2))
}
bar()
```

# Index

- \* **package**
  - document-package, [2](#)
- assign, [6](#)
- dirname, [3](#)
- display\_Rd, [5](#)
- document, [2](#), [3](#), [5](#)
- document-package, [2](#)
- get\_lines\_between\_tags, [4](#)
- help, [5](#), [6](#)
- is\_Rd\_file, [5](#)
- man, [2](#), [5](#)
- print, [6](#)
- rcmdcheck::rcmdcheck(), [4](#)
- roxygen2::roxygenize, [2](#)
- sys.call, [6](#)
- usage, [6](#)
- utils::package.skeleton, [2](#)