Package 'attachment'

May 7, 2026

Title Deal with Dependencies

Version 1.0.0

Description Manage dependencies during package development. This can retrieve all dependencies that are used in ``.R'' files in the ``R/'' directory, in ``.Rmd'' files in ``vignettes/'' directory and in 'roxygen2' documentation of functions. There is a function to update the ``DESCRIPTION'' file of your package with 'CRAN' packages or any other remote package. All functions to retrieve dependencies of ``.R'' scripts and ``.Rmd'' or ``.qmd'' files can be used independently of a package development.

License GPL-3

URL https://thinkr-open.github.io/attachment/, https://github.com/ThinkR-open/attachment

BugReports https://github.com/ThinkR-open/attachment/issues

VignetteBuilder knitr

Config/fusen/version 0.6.0

Config/Needs/website ThinkR-open/thinkrtemplate

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

Language en-US

Depends R (>= 3.4)

Imports cli, desc (>= 1.2.0), glue (>= 1.3.0), knitr (>= 1.20), magrittr (>= 1.5), rmarkdown (>= 1.10), roxygen2, stats, stringr (>= 1.3.1), utils, withr, yaml

Suggests lifecycle, renv (>= 0.8.4), rstudioapi, testthat (>= 3.0.0)

NeedsCompilation no

Author Vincent Guyader [cre, aut] (ORCID: <https://orcid.org/0000-0003-0671-9270>), Sébastien Rochette [aut] (ORCID:

<https://orcid.org/0000-0002-1565-9313>, previous maintainer),
Murielle Delmotte [aut] (ORCID:
<https://orcid.org/0000-0002-1339-2424>),
Swann Floc'hlay [aut] (ORCID: <https://orcid.org/0000-0003-1477-830X>),
ThinkR [cph, fnd]

Maintainer Vincent Guyader <vincent@thinkr.fr>

Repository CRAN

Date/Publication 2026-04-25 05:10:02 UTC

Contents

`att_amend_desc` *Amend DESCRIPTION with dependencies read from package code parsing*

Description

Amend package DESCRIPTION file with the list of dependencies extracted from R, examples, tests, vignettes files. att_to_desc_from_pkg() is an alias of att_amend_desc(), for the correspondence with `att_to_desc_from_is()`.

Usage

```
att_amend_desc(
  path = ".",
  path.n = "NAMESPACE",
  path.d = "DESCRIPTION",
```

```
  dir.r = "R",
  dir.v = "vignettes",
  dir.t = "tests",
  extra.suggests = NULL,
  pkg_ignore = NULL,
  document = TRUE,
  normalize = TRUE,
  inside_rmd = NULL,
  must.exist = TRUE,
  check_if_suggests_is_installed = TRUE,
  update.config = FALSE,
  use.config = TRUE,
  path.c = "dev/config_attachment.yaml"
)

att_to_desc_from_pkg(
  path = ".",
  path.n = "NAMESPACE",
  path.d = "DESCRIPTION",
  dir.r = "R",
  dir.v = "vignettes",
  dir.t = "tests",
  extra.suggests = NULL,
  pkg_ignore = NULL,
  document = TRUE,
  normalize = TRUE,
  inside_rmd = NULL,
  must.exist = TRUE,
  check_if_suggests_is_installed = TRUE,
  update.config = FALSE,
  use.config = TRUE,
  path.c = "dev/config_attachment.yaml"
)
```

Arguments

path	path to the root of the package directory. Default to current directory.
path.n	path to namespace file.
path.d	path to description file.
dir.r	path to directory with R scripts.
dir.v	path to vignettes directory. Set to empty (dir.v = "") to ignore.
dir.t	path to tests directory. Set to empty (dir.t = "") to ignore.
extra.suggests	vector of other packages that should be added in Suggests (pkgdown, covr for instance)
pkg_ignore	vector of packages names to ignore.
document	Run function roxygenise of roxygen2 package

normalize	Logical. Whether to normalize the DESCRIPTION file. See desc::desc_normalize()
inside_rmd	Logical or NULL. Whether the function is being called from inside a knit session, in which case the actual purl step must be delegated to an external R process. When NULL (the default), this is auto-detected via knitr::opts_knit$get("out.format").
must.exist	Logical. If TRUE then an error is given if packages do not exist within installed packages. If NA, a warning.
check_if_suggests_is_installed	Logical. Whether to require that packages in the Suggests section are installed.
update.config	logical. Should the parameters used in this call be saved in the config file of the package
use.config	logical. Should the command use the parameters from the config file to run
path.c	character Path to the yaml config file where parameters are saved

Details

Your daily use is to run att_amend_desc(), as is. You will want to run this function sometimes with some extra information like att_amend_desc(pkg_ignore = "x", update.config = TRUE) if you have to update the configuration file. Next time att_amend_desc() will use these parameters from the configuration file directly.

Value

Update DESCRIPTION file.

Examples

```
# Run on an external "dummypackage" as an example
# For your local use, you do not have to specify the `path` as below
# By default, `att_amend_desc()` will run on the current working directory

# Create a fake package for the example
tmpdir <- tempfile(pattern = "description")
dir.create(tmpdir)
file.copy(system.file("dummypackage",package = "attachment"), tmpdir,
 recursive = TRUE)
dummypackage <- file.path(tmpdir, "dummypackage")

# Update documentation and dependencies
att_amend_desc(path = dummypackage)

# You can look at the content of this external package
#' # browseURL(dummypackage)

# Update the config file with extra parameters
# We recommend that you store this code in a file in your "dev/" directory
# to run it when needed
att_amend_desc(path = dummypackage, extra.suggests = "testthat", update.config = TRUE)

# Next time, in your daily development
```

```
att_amend_desc(path = dummypackage)

# Clean after examples
unlink(tmpdir, recursive = TRUE)
```

`att_from_data` *Look for functions called in data loading code*

Description

Look for functions called in data loading code

Usage

```
att_from_data(chr)
```

Arguments

`chr`	A character vector containing the code as a string. The code should follow the pattern used for loading data with `data()`, specifying the dataset and package.

Value

A character vector containing the names of the packages from which datasets are being loaded.

Examples

```
vec_char <- 'data("starwars", package = "dplyr")'
att_from_data(vec_char)
```

`att_from_description` *Return all package dependencies from current package*

Description

Return all package dependencies from current package

Usage

```
att_from_description(
  path = "DESCRIPTION",
  dput = FALSE,
  field = c("Depends", "Imports", "Suggests")
)
```

Arguments

path	path to the DESCRIPTION file
dput	if FALSE return a vector instead of dput output
field	DESCRIPTION field to parse, Import, Suggests and Depends by default

Value

A character vector with packages names

Examples

```
dummypackage <- system.file("dummypackage", package = "attachment")
# browseURL(dummypackage)
att_from_description(path = file.path(dummypackage, "DESCRIPTION"))
```

---

att_from_examples *Get all packages called in examples from R files*

Description

Get all packages called in examples from R files

Usage

```
att_from_examples(dir.r = "R")
```

Arguments

dir.r	path to directory with R scripts.

Value

Character vector of packages called with library or require.

Examples

```
dummypackage <- system.file("dummypackage",package = "attachment")

# browseURL(dummypackage)
att_from_examples(dir.r = file.path(dummypackage,"R"))
```

att_from_namespace *return package dependencies from NAMESPACE file*

Description

return package dependencies from NAMESPACE file

Usage

```
att_from_namespace(path = "NAMESPACE", document = TRUE, clean = TRUE)
```

Arguments

path	path to NAMESPACE file
document	Run function roxygenise of roxygen2 package
clean	Logical. Whether to remove the original NAMESPACE before updating

Value

a vector

Examples

```
tmpdir <- tempfile(pattern = "namespace")
dir.create(tmpdir)
file.copy(system.file("dummypackage",package = "attachment"), tmpdir,
 recursive = TRUE)
dummypackage <- file.path(tmpdir, "dummypackage")
# browseURL(dummypackage)
att_from_namespace(path = file.path(dummypackage, "NAMESPACE"))

# Clean temp files after this example
unlink(tmpdir, recursive = TRUE)
```

att_from_rmd *Get all dependencies from a Rmd file*

Description

Get all dependencies from a Rmd file

Usage

```
att_from_rmd(
  path,
  temp_dir = tempdir(),
  warn = -1,
  encoding = getOption("encoding"),
  inside_rmd = NULL,
  inline = TRUE
)

att_from_qmd(
  path,
  temp_dir = tempdir(),
  warn = -1,
  encoding = getOption("encoding"),
  inside_rmd = NULL,
  inline = TRUE
)
```

Arguments

path	Path to a Rmd file
temp_dir	Path to temporary script from purl vignette
warn	-1 for quiet warnings with purl, 0 to see warnings
encoding	Encoding of the input file; always assumed to be UTF-8 (i.e., this argument is effectively ignored).
inside_rmd	Logical or NULL. Whether the function is being called from inside a knit session, in which case the actual purl step must be delegated to an external R process. When NULL (the default), this is auto-detected via `knitr::opts_knit$get("out.format")`.
inline	Logical. Default TRUE. Whether to explore inline code for dependencies.

Value

vector of character of packages names found in the Rmd

Examples

```
dummypackage <- system.file("dummypackage",package = "attachment")
# browseURL(dummypackage)
att_from_rmd(path = file.path(dummypackage,"vignettes/demo.Rmd"))
```

att_from_rmds *Get all packages called in vignettes folder*

Description

Get all packages called in vignettes folder

Usage

```
att_from_rmds(
  path = "vignettes",
  pattern = "*.[.](Rmd|rmd|qmd)$",
  recursive = TRUE,
  warn = -1,
  inside_rmd = NULL,
  inline = TRUE,
  folder_to_exclude = "renv"
)

att_from_qmds(
  path = "vignettes",
  pattern = "*.[.](Rmd|rmd|qmd)$",
  recursive = TRUE,
  warn = -1,
  inside_rmd = NULL,
  inline = TRUE,
  folder_to_exclude = "renv"
)
```

Arguments

path	path to directory with Rmds or vector of Rmd files
pattern	pattern to detect Rmd files
recursive	logical. Should the listing recurse into directories?
warn	-1 for quiet warnings with purl, 0 to see warnings
inside_rmd	Logical or NULL. Whether the function is being called from inside a knit session, in which case the actual purl step must be delegated to an external R process. When NULL (the default), this is auto-detected via `knitr::opts_knit$get("out.format")`.
inline	Logical. Default TRUE. Whether to explore inline code for dependencies.
folder_to_exclude	Folder to exclude during scan to detect packages 'renv' by default

Value

Character vector of packages called with library or require. When the directory contains "vignettes" in its path, `knitr` is always added and the vignette engine is inferred from the files actually present: `rmarkdown` is added when `.Rmd` files are found, `quarto` when `.qmd` files are found (both when the directory mixes the two). If the directory is empty, `rmarkdown` is added as a safe default.

Examples

```
dummypackage <- system.file("dummypackage",package = "attachment")
# browseURL(dummypackage)
att_from_rmds(path = file.path(dummypackage,"vignettes"))
```

---

att_from_rscript *Look for functions called with :: and library/requires in one script*

Description

Look for functions called with :: and library/requires in one script

Usage

```
att_from_rscript(path, encoding = getOption("encoding"))
```

Arguments

`path`	path to R script file
`encoding`	Encoding passed to `readLines()` when reading `path`. Defaults to `getOption("encoding")` so the system locale is respected (important on Windows where scripts are often Latin-1 / Windows-1252).

Details

Uses the R parser to walk the syntax tree so that occurrences of `pkg::fun` or `library()/require()/requireNamespace()/` inside string literals or comments are ignored. Named arguments such as `library(package = "pkg")` are supported, as are fully-qualified forms like `base::library(pkg)` or `methods::getFromNamespace(fn, "pkg")`. Introspection helpers such as `packageVersion()`, `getNamespace()`, `asNamespace()`, and `attachNamespace()` are **not** treated as dependency introducers, because they are commonly used for version or feature checks on packages that may or may not be required at runtime.

If the file cannot be parsed as valid R (syntax error, corrupt encoding, etc.), the function falls back to a regex-based detector and emits a `warning()` naming the file so users can investigate.

Value

a vector

Examples

```
dummypackage <- system.file("dummypackage",package = "attachment")
# browseURL(dummypackage)

att_from_rscript(path = file.path(dummypackage,"R","my_mean.R"))
```

`att_from_rscripts` *Look for functions called with* `::` *and library/requires in folder of scripts*

Description

Look for functions called with `::` and library/requires in folder of scripts

Usage

```
att_from_rscripts(
  path = "R",
  pattern = "*.[.](r|R)$",
  recursive = TRUE,
  folder_to_exclude = "renv",
  encoding = getOption("encoding")
)
```

Arguments

`path`	directory with R scripts inside or vector of R scripts
`pattern`	pattern to detect R script files
`recursive`	logical. Should the listing recurse into directories?
`folder_to_exclude`	Folder to exclude during scan to detect packages. 'renv' by default.
`encoding`	Encoding passed to `readLines()` when reading `path`. Defaults to `getOption("encoding")` so the system locale is respected (important on Windows where scripts are often Latin-1 / Windows-1252).

Value

vector of character of packages names found in the R script

Examples

```
dummypackage <- system.file("dummypackage",package = "attachment")
# browseURL(dummypackage)

att_from_rscripts(path = file.path(dummypackage, "R"))
att_from_rscripts(path = list.files(file.path(dummypackage, "R"), full.names = TRUE))
```

`att_to_desc_from_is` *Amend DESCRIPTION with dependencies from imports and suggests package list*

Description

Amend DESCRIPTION with dependencies from imports and suggests package list

Usage

```
att_to_desc_from_is(
  path.d = "DESCRIPTION",
  imports = NULL,
  suggests = NULL,
  check_if_suggests_is_installed = TRUE,
  normalize = TRUE,
  must.exist = TRUE
)
```

Arguments

`path.d`	path to description file.
`imports`	character vector of package names to add in Imports section
`suggests`	character vector of package names to add in Suggests section
`check_if_suggests_is_installed`	Logical. Whether to require that packages in the Suggests section are installed.
`normalize`	Logical. Whether to normalize the DESCRIPTION file. See `desc::desc_normalize()`
`must.exist`	Logical. If TRUE then an error is given if packages do not exist within installed packages. If NA, a warning.

Details

`must.exist` is better set to `TRUE` during package development. This stops the process when a package does not exists on your system. This avoids check errors with typos in package names in DESCRIPTION. When used in CI to discover dependencies, for a bookdown for instance, you may want to set to `FALSE` (no message at all) or `NA` (warning for not installed).

Value

Fill in Description file

Examples

```
tmpdir <- tempfile(pattern = "descfromis")
dir.create(tmpdir)
file.copy(system.file("dummypackage",package = "attachment"), tmpdir,
 recursive = TRUE)
dummypackage <- file.path(tmpdir, "dummypackage")
# browseURL(dummypackage)
att_to_desc_from_is(path.d = file.path(dummypackage, "DESCRIPTION"),
imports = c("magrittr", "attachment"), suggests = c("knitr"))

# In combination with other functions
att_to_desc_from_is(path.d = file.path(dummypackage, "DESCRIPTION"),
imports = att_from_rscripts(file.path(dummypackage, "R")),
suggests = att_from_rmds(file.path(dummypackage, "vignettes")))

# Clean temp files after this example
unlink(tmpdir, recursive = TRUE)
```

---

create_dependencies_file

Create the list of instructions to install dependencies from a DESCRIPTION file

---

Description

Outputs the list of instructions and a "dependencies.R" file with instructions in the "inst/" directory

Usage

```
create_dependencies_file(
  path = "DESCRIPTION",
  field = c("Depends", "Imports"),
  to = "inst/dependencies.R",
  open_file = TRUE,
  ignore_base = TRUE,
  install_only_if_missing = FALSE
)
```

Arguments

path	path to the DESCRIPTION file
field	DESCRIPTION field to parse, "Import" and "Depends" by default. Can add "Suggests"
to	path where to save the dependencies file. Set to "inst/dependencies.R" by default. Set to NULL if you do not want the file, but only the instructions as a list of character.
open_file	Logical. Open the file created in an editor

ignore_base Logical. Whether to ignore package coming with base, as they cannot be installed (default TRUE)

install_only_if_missing
Logical Modify the installation instructions to check, beforehand, if the packages are missing . (default FALSE)

Value

List of R instructions to install all dependencies from a DESCRIPTION file. Side effect: creates a R file containing these instructions.

Examples

```
# Create a fake package
tmpdir <- tempfile(pattern = "depsfile")
dir.create(tmpdir)
file.copy(system.file("dummypackage",package = "attachment"), tmpdir,
          recursive = TRUE)
dummypackage <- file.path(tmpdir, "dummypackage")

# Create the dependencies commands but no file
create_dependencies_file(
  path = file.path(dummypackage,"DESCRIPTION"),
  to = NULL,
  open_file = FALSE)

# Create the dependencies files in the package
create_dependencies_file(
  path = file.path(dummypackage,"DESCRIPTION"),
  to = file.path(dummypackage, "inst/dependencies.R"),
  open_file = FALSE)
list.files(file.path(dummypackage, "inst"))
# browseURL(dummypackage)

# Clean temp files after this example
unlink(tmpdir, recursive = TRUE)
```

---

create_renv_for_dev *Create reproducible environments for your R projects with* renv

---

Description

[Experimental]

Tool to create and maintain renv.lock files. The idea is to have 2 distinct files, one for development and the other for deployment. Indeed, although packages like *attachment* or *pkgload* must be installed to develop, they are not necessary in your project, package or Shiny application.

Usage

```
create_renv_for_dev(
  path = ".",
  dev_pkg = "_default",
  folder_to_include = c("dev", "data-raw"),
  folder_to_exclude = c("renv"),
  output = "renv.lock",
  install_if_missing = TRUE,
  document = TRUE,
  pkg_ignore = NULL,
  check_if_suggests_is_installed = TRUE,
  ...
)

create_renv_for_prod(
  path = ".",
  output = "renv.lock.prod",
  dev_pkg = "remotes",
  check_if_suggests_is_installed = FALSE,
  folder_to_include = NULL,
  ...
)
```

Arguments

path	Path to your current package source folder
dev_pkg	Vector of packages you need for development. Use `_default` (with underscore before to avoid confusing with a package name), to use the default list. Use `NULL` for no extra package. Use `attachment:::extra_dev_pkg` for the list.
folder_to_include	Folder to scan to detect development packages
folder_to_exclude	Folder to exclude during scan to detect packages.'renv' by default
output	Path and name of the file created, default is `./renv.lock`
install_if_missing	Logical. Install missing packages. `TRUE` by default
document	Logical. Whether to run `att_amend_desc()` before detecting packages in DESCRIPTION.
pkg_ignore	Vector of packages to ignore from being discovered in your files. This does not prevent them to be in "renv.lock" if they are recursive dependencies.
check_if_suggests_is_installed	Logical. Whether to require that packages in the Suggests section are installed.
...	Other arguments to pass to `renv::snapshot()`

Value

a renv.lock file

Examples

```
## Not run:
# Writes a renv.lock a file in the user directory
create_renv_for_dev()
create_renv_for_dev(dev_pkg = "attachment")
create_renv_for_prod()

## End(Not run)
```

`find_remotes` *Proposes values for Remotes field for DESCRIPTION file based on your installation*

Description

Proposes values for Remotes field for DESCRIPTION file based on your installation

Usage

```
find_remotes(pkg)
```

Arguments

pkg	Character. Packages to test for potential non-CRAN installation

Value

List of all non-CRAN packages and code to add in Remotes field in DESCRIPTION. NULL otherwise.

Examples

```
# Find from vector of packages
find_remotes(pkg = c("attachment", "desc", "glue"))

# Find from Description file
dummypackage <- system.file("dummypackage", package = "attachment")
att_from_description(
path = file.path(dummypackage, "DESCRIPTION")) %>%
find_remotes()

## Not run:
# For the current package directory
att_from_description() %>% find_remotes()

## End(Not run)


# For a specific package name
```

```
find_remotes("attachment")

# Find remotes from all installed packages
find_remotes(list.dirs(.libPaths(), full.names = FALSE, recursive = FALSE))
```

`install_from_description` *Install missing package from DESCRIPTION*

Description

Install missing package from DESCRIPTION

Usage

```
install_from_description(
  path = "DESCRIPTION",
  field = c("Depends", "Imports", "Suggests"),
  ...
)
```

Arguments

`path`	path to the DESCRIPTION file
`field`	DESCRIPTION fields to parse, "Depends", "Imports", "Suggests" by default
`...`	Arguments to be passed to `utils::install.packages()`

Value

Used for side effect. Installs R packages from DESCRIPTION file if missing.

Examples

```
## Not run:
# This will install packages on your system
dummypackage <- system.file("dummypackage", package = "attachment")
# browseURL(dummypackage)

install_from_description(path = file.path(dummypackage, "DESCRIPTION"))

## End(Not run)
```

install_if_missing *install packages if missing*

Description

install packages if missing

Usage

```
install_if_missing(to_be_installed, ...)
```

Arguments

to_be_installed	a character vector containing required packages names
...	Arguments to be passed to utils::install.packages()

Value

Used for side effect. Install missing packages from the character vector input.

Examples

```
## Not run:
# This will install packages on your system
install_if_missing(c("dplyr", "fcuk", "rusk"))

## End(Not run)
```

set_remotes_to_desc *Add Remotes field to DESCRIPTION based on your local installation*

Description

Add Remotes field to DESCRIPTION based on your local installation

Usage

```
set_remotes_to_desc(path.d = "DESCRIPTION", stop.local = FALSE, clean = TRUE)
```

Arguments

path.d	path to description file.
stop.local	Logical. Whether to stop if package was installed from local source. Message otherwise.
clean	Logical. Whether to clean all existing remotes before run.

Value

Used for side effect. Adds Remotes field in DESCRIPTION file.

Examples

```
tmpdir <- tempfile(pattern = "setremotes")
dir.create(tmpdir)
file.copy(system.file("dummypackage", package = "attachment"), tmpdir,
 recursive = TRUE)
dummypackage <- file.path(tmpdir, "dummypackage")
# Add remotes field if there are Remotes locally
att_amend_desc(dummypackage) %>%
  set_remotes_to_desc()

# Clean temp files after this example
unlink(tmpdir, recursive = TRUE)

## Not run:
# For your current package
att_amend_desc() %>%
  set_remotes_to_desc()

## End(Not run)
```

Index