# Package ‘pkgload’

February 3, 2026

**Title** Simulate Package Installation and Attach

**Version** 1.5.0

**Description** Simulates the process of installing a package and then attaching it. This is a key part of the 'devtools' package as it allows you to rapidly iterate while developing a package.

**License** MIT + file LICENSE

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

**BugReports** https://github.com/r-lib/pkgload/issues

**Depends** R (>= 3.4.0)

**Imports** cli (>= 3.3.0), desc, fs, glue, lifecycle, methods, pkgbuild, processx, rlang (>= 1.1.1), rprojroot, utils

**Suggests** bitops, jsonlite, mathjaxr, pak, Rcpp, remotes, rstudioapi, testthat (>= 3.2.1.1), usethis, withr

**Config/Needs/website** tidyverse/tidytemplate, ggplot2

**Config/testthat/edition** 3

**Config/testthat/parallel** TRUE

**Config/testthat/start-first** dll

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Hadley Wickham [aut],
Winston Chang [aut],
Jim Hester [aut],
Lionel Henry [aut, cre],
Posit Software, PBC [cph, fnd],
R Core team [ctb] (Some namespace and vignette code extracted from base R)

**Maintainer** Lionel Henry <lionel@posit.co>

**Repository** CRAN

**Date/Publication** 2026-02-03 06:10:46 UTC

# Contents



---

`dev_example` *Run a examples for an in-development function.*

---

## Description

`dev_example` is a replacement for `example`. `run_example` is a low-level function that takes a path to an Rd file.

## Usage

```
dev_example(topic, quiet = FALSE)

run_example(
  path,
  run_donttest = FALSE,
  run_dontrun = FALSE,
  env = new.env(parent = globalenv()),
  quiet = FALSE,
  macros = NULL,
  run,
  test
)
```

## Arguments

| | |
|---|---|
| `topic` | Name or topic (or name of Rd) file to run examples for |
| `quiet` | If `TRUE`, does not echo code to console. |
| `path` | Path to `.Rd` file |
| `run_donttest` | if `TRUE`, do run `\donttest` sections in the Rd files. |

| | |
|---|---|
| run_dontrun | if TRUE, do run \dontrun sections in the Rd files. |
| env | Environment in which code will be run. |
| macros | Custom macros to use to parse the .Rd file. See the macros argument of tools::parse_Rd(). If NULL, then the tools::Rd2ex() (and tools::parse_Rd()) default is used. |
| run, test | Deprecated, see run_dontrun and run_donttest above. |

## Examples

```
## Not run:
# Runs installed example:
library("ggplot2")
example("ggplot")

# Runs development example:
dev_example("ggplot")

## End(Not run)
```

---

dev_help *In-development help for package loaded with devtools*

---

## Description

dev_help() searches for source documentation provided in packages loaded by devtools. To improve performance, the .Rd files are parsed to create to index once, then cached. Use dev_topic_index_reset() to clear that index. You can manually retrieve the index for a local package with dev_topic_index().

## Usage

```
dev_help(
  topic,
  dev_packages = NULL,
  stage = "render",
  type = getOption("help_type")
)

dev_topic_find(topic, dev_packages = NULL)

dev_topic_index(path = ".")

dev_topic_index_reset(pkg_name)
```

## Arguments

| | |
|---|---|
| topic | name of help to search for. |
| dev_packages | A character vector of package names to search within. If NULL, defaults to all packages loaded by devtools. |

| | |
|---|---|
| stage | at which stage ("build", "install", or "render") should \\Sexpr macros be executed? This is only important if you're using \\Sexpr macro's in your Rd files. |
| type | of html to produce: "html" or "text". Defaults to your default documentation type. |
| path | Path to package. |
| pkg_name | Name of package. |

## Examples

```
## Not run:
library("ggplot2")
help("ggplot") # loads installed documentation for ggplot

load_all("ggplot2")
dev_help("ggplot") # loads development documentation for ggplot

## End(Not run)
```

---

# help *Drop-in replacements for help and ? functions*

---

## Description

The ? and help functions are replacements for functions of the same name in the utils package. They are made available when a package is loaded with load_all().

## Usage

```
# help(topic, package = NULL, ...)

# ?e2
# e1?e2
```

## Arguments

| | |
|---|---|
| topic | A name or character string specifying the help topic. |
| package | A name or character string specifying the package in which to search for the help topic. If NULL, search all packages. |
| ... | Additional arguments to pass to utils::help(). |
| e1 | First argument to pass along to utils::¿'. |
| e2 | Second argument to pass along to utils::¿'. |

## Details

The ? function is a replacement for utils::?() from the utils package. It will search for help in devtools-loaded packages first, then in regular packages.

The help function is a replacement for utils::help() from the utils package. If package is not specified, it will search for help in devtools-loaded packages first, then in regular packages. If package is specified, then it will search for help in devtools-loaded packages or regular packages, as appropriate.

## Examples

```
## Not run:
# This would load devtools and look at the help for load_all, if currently
# in the devtools source directory.
load_all()
?load_all
help("load_all")

## End(Not run)

# To see the help pages for utils::help and utils::`?`:
help("help", "utils")
help("?", "utils")

## Not run:
# Examples demonstrating the multiple ways of supplying arguments
# NB: you can't do pkg <- "ggplot2"; help("ggplot2", pkg)
help(lm)
help(lm, stats)
help(lm, 'stats')
help('lm')
help('lm', stats)
help('lm', 'stats')
help(package = stats)
help(package = 'stats')
topic <- "lm"
help(topic)
help(topic, stats)
help(topic, 'stats')

## End(Not run)
```

---

# inst *Get the installation path of a package*

---

## Description

Given the name of a package, this returns a path to the installed copy of the package, which can be passed to other devtools functions.

### Usage

```
inst(name)
```

### Arguments

| | |
|---|---|
| name | the name of a package. |

### Details

It searches for the package in `.libPaths()`. If multiple dirs are found, it will return the first one.

### Examples

```
inst("pkgload")
inst("grid")
```

---

## is_dev_package *Is the package currently under development?*

---

### Description

Returns `TRUE` or `FALSE` depending on if the package has been loaded by **pkgload**.

### Usage

```
is_dev_package(name)
```

### Arguments

| | |
|---|---|
| name | the name of a package. |

---

## load_all *Load complete package*

---

### Description

`load_all()` loads a package. It roughly simulates what happens when a package is installed and loaded with `library()`, without having to first install the package. It:

- Loads all data files in `data/`. See `load_data()` for more details.
- Sources all R files in the R directory, storing results in environment that behaves like a regular package namespace. See `load_code()` for more details.
- Adds a shim from `system.file()` to `shim_system.file()` in the imports environment of the package. This ensures that `system.file()` works with both development and installed packages despite their differing directory structures.

- Adds shims from `help()` and `?` to `shim_help()` and `shim_question()` to make it easier to preview development documentation.
- Compiles any C, C++, or Fortran code in the `src/` directory and connects the generated DLL into R. See `pkgbuild::compile_dll()` for more details.
- Loads any compiled translations in `inst/po`.
- Runs `.onAttach()`, `.onLoad()` and `.onUnload()` functions at the correct times.
- If you use **testthat**, will load all test helpers so you can access them interactively. devtools sets the `DEVTOOLS_LOAD` environment variable to the package name to let you check whether the helpers are run during package loading.

`is_loading()` returns `TRUE` when it is called while `load_all()` is running. This may be useful e.g. in `.onLoad` hooks. A package loaded with `load_all()` can be identified with `is_dev_package()`.

## Usage

```
load_all(
  path = ".",
  reset = TRUE,
  compile = NA,
  attach = TRUE,
  export_all = TRUE,
  export_imports = export_all,
  helpers = export_all,
  attach_testthat = uses_testthat(path),
  quiet = NULL,
  recompile = FALSE,
  warn_conflicts = TRUE,
  debug = TRUE
)

is_loading(pkg = NULL)
```

## Arguments

| | |
|---|---|
| `path` | Path to a package, or within a package. |
| `reset` | **[Deprecated]** This is no longer supported because preserving the namespace requires unlocking its environment, which is no longer possible in recent versions of R. |
| `compile` | If `TRUE` always recompiles the package; if `NA` recompiles if needed (as determined by `pkgbuild::needs_compile()`); if `FALSE`, never recompiles. |
| `attach` | Whether to attach a package environment to the search path. If `FALSE` `load_all()` behaves like `loadNamespace()`. If `TRUE` (the default), it behaves like `library()`. If `FALSE`, the `export_all`, `export_imports`, and `helpers` arguments have no effect. |
| `export_all` | If `TRUE` (the default), export all objects. If `FALSE`, export only the objects that are listed as exports in the NAMESPACE file. |

export_imports If TRUE (the default), export all objects that are imported by the package. If FALSE export only objects defined in the package.

helpers if TRUE loads **testthat** test helpers.

attach_testthat
If TRUE, attach **testthat** to the search path, which more closely mimics the environment within test files.

quiet if TRUE suppresses output from this function.

recompile DEPRECATED. force a recompile of DLL from source code, if present. This is equivalent to running pkgbuild::clean_dll() before load_all()

warn_conflicts If TRUE, issues a warning if a function in the global environment masks a function in the package. This can happen when you accidentally source a .R file, rather than using load_all(), or if you define a function directly in the R console. This is frustrating to debug, as it feels like the changes you make to the package source aren't having the expected effect.

debug If TRUE (the default), then the build runs without optimisation (-O0) and with debug symbols (-g). See pkgbuild::compile_dll() for details.

pkg If supplied, is_loading() only returns TRUE if the package being loaded is pkg.

## Differences to regular loading

load_all() tries its best to reproduce the behaviour of loadNamespace() and library(). However it deviates from normal package loading in several ways.

- load_all() doesn't install the package to a library, so system.file() doesn't work. pkgload fixes this for the package itself installing a shim, shim_system.file(). However, this shim is not visible to third party packages, so they will fail if they attempt to find files within your package. One potential workaround is to use fs::path_package() instead of system.file(), since that understands the mechanisms that devtools uses to load packages.
- load_all() loads all packages referenced in Imports at load time, but loadNamespace() and library() only load package dependencies as they are needed.
- load_all() copies all objects (not just the ones listed as exports) into the package environment. This is useful during development because it makes internal objects easy to access. To export only the objects listed as exports, use export_all = FALSE. This more closely simulates behavior when loading an installed package with library(), and can be useful for checking for missing exports.

## Controlling the debug compiler flags

load_all() delegates to pkgbuild::compile_dll() to perform the actual compilation, during which by default some debug compiler flags are appended. If you would like to produce an optimized build instead, you can opt out by either using debug = FALSE, setting the pkg.build_extra_flags option to FALSE, or setting the PKG_BUILD_EXTRA_FLAGS environment variable to FALSE. For further details see the Details section in pkgbuild::compile_dll().

## Examples

```
## Not run:
# Load the package in the current directory
load_all("./")

# Running again loads changed files
load_all("./")

# With export_all=FALSE, only objects listed as exports in NAMESPACE
# are exported
load_all("./", export_all = FALSE)

## End(Not run)
```

---

`load_code` *Load R code.*

---

## Description

Sources all `.R/.r` files in the `R/` directory, storing results into the package namespace.

## Usage

```
load_code(path = ".", quiet = NULL)
```

## Arguments

| | |
|---|---|
| `path` | Path to a package, or within a package. |
| `quiet` | if `TRUE` suppresses output from this function. |

---

`load_data` *Load data.*

---

## Description

Loads all `.RData` files in the data subdirectory.

## Usage

```
load_data(path = ".")
```

## Arguments

| | |
|---|---|
| `path` | Path to a package, or within a package. |

## load_dll *Load a compiled DLL*

### Description

Load a compiled DLL

### Usage

```
load_dll(path = ".")
```

### Arguments

| | |
|---|---|
| path | Path to a package, or within a package. |

## packages *Helper functions for working with development packages.*

### Description

All functions search recursively up the directory tree from the input path until they find a DESCRIPTION file.

### Usage

```
pkg_path(path = ".")

pkg_name(path = ".")

pkg_desc(path = ".")

pkg_version(path = ".")

pkg_version_raw(path = ".")

pkg_ns(path = ".")
```

### Arguments

| | |
|---|---|
| path | Path to a package, or within a package. |

### Functions

- `pkg_path()`: Return the normalized package path.
- `pkg_name()`: Return the package name.
- `pkg_desc()`: Return the package DESCRIPTION as a `desc::desc()` object.
- `pkg_version()`: Return the parsed package version.
- `pkg_version_raw()`: Return the raw package version (as a string).
- `pkg_ns()`: Return the package namespace.

---

## `package_file` *Find file in a package.*

---

### Description

It always starts by finding by walking up the path until it finds the root directory, i.e. a directory containing `DESCRIPTION`. If it cannot find the root directory, or it can't find the specified path, it will throw an error.

### Usage

```
package_file(..., path = ".")
```

### Arguments

| | |
|---|---|
| `...` | Components of the path. |
| `path` | Place to start search for package directory. |

### Examples

```
## Not run:
package_file("figures", "figure_1")

## End(Not run)
```

## system.file *Replacement version of system.file*

### Description

This function is meant to intercept calls to base::system.file(), so that it behaves well with packages loaded by devtools. It is made available when a package is loaded with load_all().

### Usage

```
shim_system.file(..., package = "base", lib.loc = NULL, mustWork = FALSE)
```

### Arguments

| | |
|---|---|
| ... | character vectors, specifying subdirectory and file(s) within some package. The default, none, returns the root of the package. Wildcards are not supported. |
| package | a character string with the name of a single package. An error occurs if more than one package name is given. |
| lib.loc | a character vector with path names of R libraries. See 'Details' for the meaning of the default value of NULL. |
| mustWork | logical. If TRUE, an error is given if there are no matching files. |

### Details

When system.file is called from the R console (the global environment), this function detects if the target package was loaded with load_all(), and if so, it uses a customized method of searching for the file. This is necessary because the directory structure of a source package is different from the directory structure of an installed package.

When a package is loaded with load_all, this function is also inserted into the package's imports environment, so that calls to system.file from within the package namespace will use this modified version. If this function were not inserted into the imports environment, then the package would end up calling base::system.file instead.

## unload *Unload a package*

### Description

unload() attempts to cleanly unload a package, including unloading its namespace, deleting S4 class definitions and unloading any loaded DLLs. Unfortunately S4 classes are not really designed to be cleanly unloaded, and so we have to manually modify the class dependency graph in order for it to work - this works on the cases for which we have tested but there may be others. Similarly, automated DLL unloading is best tested for simple scenarios (particularly with

useDynLib(pkgname) and may fail in other cases. If you do encounter a failure, please file a bug report at https://github.com/r-lib/pkgload/issues.

unregister() is a gentler version of unload() which removes the package from the search path, unregisters methods, and unregisters the namespace. It doesn't unload the namespace or its DLL to keep it in working order in case of dangling references.

## Usage

```
unload(package = pkg_name(), quiet = FALSE)

unregister(package = pkg_name())
```

## Arguments

| | |
|---|---|
| package | package name. |
| quiet | if TRUE suppresses output from this function. |

## Examples

```
## Not run:
# Unload package that is in current directory
unload()

# Unload package that is in ./ggplot2/
unload(pkg_name("ggplot2/"))

library(ggplot2)
# unload the ggplot2 package directly by name
unload("ggplot2")

## End(Not run)
```

# Index