

Package: adorn (via r-universe)

October 26, 2026

Title A Framework for Writing and Applying Function Decorators

Version 0.0.0.9000

Description A framework for creating function decorators in R.
Decorators can be applied to functions and to S3, S4, S7 and R6
methods.

License MIT + file LICENSE

URL <https://dyfanjones.github.io/adorn/>,
<https://github.com/DyfanJones/adorn>

BugReports <https://github.com/DyfanJones/adorn/issues>

Depends R (>= 4.4.0)

Imports methods, S7 (>= 0.2.0), utils

Suggests litedown, R6, testthat (>= 3.0.0)

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

Encoding UTF-8

Roxygen list(markdown = TRUE)

Collate 'context.R' 'decorator.R' 'decorate.R'

VignetteBuilder litedown

Repository <https://dyfanjones.r-universe.dev>

Date/Publication 2026-10-26 18:30:23 UTC

RemoteUrl <https://github.com/DyfanJones/adorn>

RemoteRef HEAD

RemoteSha b890b37326c2a03f440c12974ddbe788f86f9aa5

Contents

apply-decorator	2
decorate	2
decorator	4
decorator_context	6
undecorate	6

Index

7

apply-decorator	<i>Apply decorators to a function</i>
-----------------	---------------------------------------

Description

%% reads left to right: in a %% b %% f, a is the outermost decorator and f the innermost function. A plain function(f) that returns a function also works as a decorator on the left-hand side.

Usage

```
lhs %% rhs
```

Arguments

lhs	A decorator.
rhs	A function, or another decorator to compose with.

Value

The decorated function, or a composed decorator if rhs is itself a decorator.

Examples

```
loud <- decorator(function(f, ...) toupper(f(...))) %% function() "hi"
loud()
```

decorate	<i>Decorate a function or an existing method</i>
----------	--

Description

decorate() applies a decorator to a target: a function, or a method that already exists on an S3, S4, S7 or R6 class. Use %% to decorate a function you are defining, and decorate() for code you did not write.

Usage

```
decorate(target, decorator, ...)

decorate(target, signature) <- value
```

Arguments

target	What to decorate; see the table above.
decorator	A decorator, or a stack of them built with %%%.
...	Arguments for specific kinds of target: • class: for S3, the class name as a string. For S7, the class or signature, as in <code>S7::method()</code>. For R6, the class generator. • signature: for S4, the signature, as in <code>methods::setMethod()</code>. • envir: for S3, the environment to find the method and generic in. Defaults to the caller's. • where: for S4, the environment to store the method in. Defaults to the top-level environment of the caller. For R6, the part of the class that holds the method: "public" (the default), "private" or "active".
signature	For the replacement form, the S7 class, S4 signature, S3 class name, or R6 method name(s), as above.
value	For the replacement form, the decorator to apply.

Details

target	Example	Extra argument
a function	<code>decorate(fn, d)</code> , the same as <code>d %%% fn</code>	
the name of an S3 generic	<code>decorate("print", d, class = "money")</code>	class
an S4 generic	<code>decorate(area, d, signature = "Square")</code>	signature
an S7 generic	<code>decorate(speak, d, class = Dog)</code>	class
the name of an R6 method	<code>decorate("add", d, class = Counter)</code>	class

For S3 and R6 the method is named by a string, and class gives the class or R6 generator. For S4 and S7 the generic is the target. The replacement form (see below) works for all four systems.

Value

For a function, the decorated function. For R6, the class generator, invisibly. For the others, the decorated method, invisibly; the method is replaced in place.

R6 classes

`decorate("name", decorator, class = Generator)` replaces the method in the generator, so objects created afterwards use it. Objects that already exist keep the old method, because R6 copies methods into each object when it is created. An inherited method is copied into Generator and decorated there; the parent class is unchanged. Give several names to decorate several methods.

Replacement form

`decorate(target, class) <- decorator` mirrors S7's `method(generic, class) <- fn` and works for all four systems:

```

decorate(speak, Dog) <- shout      # S7
decorate(area, "Square") <- shout  # S4
decorate(print, "money") <- shout  # S3, with the generic function
decorate(Counter, "add") <- shout  # R6, public methods

```

R does not allow a string as the target of a replacement call, so for S3 pass the generic function. Its name is read from its `UseMethod()` call, so the generic must call `UseMethod()` directly. For R6 the form reaches public methods only; use the explicit form for where.

Examples

```

# A function
shout <- decorator(function(f, ...) toupper(f(...)))
greet <- decorate(function(who) paste("hello", who), shout)
greet("world")

# An existing S3 method
area <- function(x, ...) UseMethod("area")
area.square <- function(x, ...) x$side^2
decorate("area", shout, class = "square")
area(structure(list(side = 3), class = "square"))

# An R6 method
if (requireNamespace("R6", quietly = TRUE)) {
  Greeter <- R6::R6Class("Greeter", public = list(
    hi = function(who) paste("hello", who)
  ))
  decorate("hi", shout, class = Greeter)
  Greeter$new()$hi("world")
}

```

decorator

Create a decorator

Description

`decorator()` turns an ordinary function into a decorator. The first argument of wrapper receives the function being decorated. There are two styles, chosen by whether wrapper has a `...` argument:

Usage

```
decorator(wrapper, name = NULL)
```

Arguments

wrapper	A function, as described above.
name	Name shown when printing the decorator and in <code>decorators()</code> . By default it is taken from how the decorator is written when applied with <code>%%</code> (timed <code>%%</code> <code>f</code> gives "timed", and <code>retry(times = 5) %%</code> <code>f</code> gives "retry"). Set it to override that; "decorator" is the last resort.

Details

- **Call style**, `function(f, ..., <params>)`: the body runs on every call. ... holds the arguments of that call; forward them with `f(...)`.
- **Factory style**, `function(f, <params>)`: the body runs once, when the function is decorated, and returns the replacement function. Use this when the decorator needs state, such as a cache.

Arguments after ... (call style) or after `f` (factory style) are decorator parameters. Set them by calling the decorator: `retry(times = 5)`, for a decorator named `retry`.

If your decorator stores the result of `f(...)` before returning it, use `withVisible()` to keep functions that return `invisible()` quiet.

Whatever the style, the decorated function keeps the formal arguments of the original, so argument matching, `missing()`, defaults and introspection all behave as before.

Value

A decorator: a function that decorates a function when called with one, and returns a configured decorator when called with parameters.

Examples

```
# Call style: runs on every call
shout <- decorator(function(f, ...) toupper(f(...)))

# Call style with a parameter
repeated <- decorator(function(f, ..., times = 2) {
  rep(f(...), times)
})

# Factory style: keeps state between calls
counted <- decorator(function(f) {
  n <- 0
  function(...) {
    n <- n + 1
    message("call ", n)
    f(...)
  }
})

greet <- shout %% function(who) paste("hello", who)
greet("world")

ones <- repeated(times = 3) %% function() 1
ones()
```

decorator_context	<i>Information about the function being decorated</i>
-------------------	---

Description

Call this from inside a decorator to find out which function it is currently wrapping.

Usage

```
decorator_context()
```

Value

A list with name (the name the function is bound to, or "<anonymous>"), decorator (the decorator's name, see [decorator\(\)](#)) and fn (the function being wrapped).

Examples

```
who <- decorator(function(f, ...) {
  message("calling ", decorator_context()$name)
  f(...)
})
hello <- who %%% function() "hi"
hello()
```

undecorate	<i>Inspect a decorated function</i>
------------	-------------------------------------

Description

layers() is an alias for decorators(), and strip() for undecorate().

Usage

```
undecorate(f)
```

```
decorators(f)
```

Arguments

f	A function.
---	-------------

Value

undecorate() returns the original, undecorated function (or f itself if it is not decorated). decorators() returns the names of the decorators applied, outermost first.

Index

`apply-decorator`, [2](#)

`decorate`, [2](#)

`decorate<- (decorate)`, [2](#)

`decorator`, [4](#)

`decorator()`, [6](#)

`decorator_context`, [6](#)

`decorators (undecorate)`, [6](#)

`decorators()`, [4](#)

`methods::setMethod()`, [3](#)

`S7::method()`, [3](#)

`undecorate`, [6](#)