

Package ‘withdots’

May 8, 2026

Type Package

Title Put ... in a Function's Argument List

Version 0.1.1

Description Adds ... to a function's argument list so that it can tolerate non-matching arguments.

License MIT + file LICENSE

URL <https://github.com/NikKrieger/withdots>

BugReports <https://github.com/NikKrieger/withdots/issues>

Suggests rlang

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation no

Author Nikolas Ivan Krieger [aut, cre]

Maintainer Nikolas Ivan Krieger <nk@case.edu>

Repository CRAN

Date/Publication 2023-04-28 18:10:02 UTC

Contents

withdots	2
Index	6

withdots

Give a [function](#) ... if it does not have it

Description

Adds ... to a [closure](#)'s [args](#) if it does not have it already.

Usage

```
withdots(f)
```

Arguments

`f` A [function](#). See **Handling of [primitives](#)** in case `f` is [primitive](#).

Details

If `f` already has ... in its [args](#), then it is returned with no changes. Otherwise, ... is added to `f`'s [formals](#) and then `f` is returned. See **Handling of [primitives](#)** below.

Value

If `f` has ... in its [args](#), then `f`.

Otherwise, a [closure](#): a tweaked version of `f`, whose only differences are:

1. ... has been appended to the end of its [formals](#), and
2. any [srcref attribute](#) has been removed (see **Why the [srcref attribute](#) is removed** below).

How ... is added to [closures](#)

These are the steps that `withdots()` takes **only** if `f` is a [closure](#) without ... in its [formals](#):

1. [attributes](#)(`f`) are temporarily saved and set aside.
2. If there is a [srcref attribute](#) among the set-aside [attributes](#)(`f`), it is removed (see **Why the [srcref attribute](#) is removed** below).
3. ... is added to the [formals](#) of `f` using `formals<-`.
4. The remaining set-aside [attributes](#) are added back to `f` with `attributes<-`.
5. `f` is returned.

Handling of [primitives](#)

If `f` is [primitive](#) and already has ... in its [args](#) (e.g., `c()`, `rep()`, `max()`), then it is returned as is.

If `f` is [primitive](#) and does **not** have ... in its [args](#), then an error will be thrown. The user can bypass this error by processing `f` with `rlang::as_closure()` before passing it to `withdots()`. **However, keep in mind that the argument matching behavior of the resulting [closure](#) may be different from what is expected, since [primitives](#) may use nonstandard argument matching.**

Why the `srcref` attribute is removed

Many `functions`—including those created with `function()`—have a `srcref` attribute. When a `function` is `printed`, `print.function()` relies on this `attribute` by default to depict the `function`'s `formals` and `body`.

`withdots()` adds `...` via `formals<-`, which expressly drops `attributes` (see its [documentation page](#)). To prevent this loss, `withdots()` sets `attributes(f)` aside at the beginning and re-attaches them to `f` at the end. Normally, this would re-attach the original `f`'s `srcref` attribute to the new `f`, making it so that the newly added `...` would not be depicted when the new `f` is `printed`. For this reason, the old `srcref` attribute is dropped, and only the remaining `attributes` are re-attached to the new `f`.

Observe what would happen during `printing` if **all** original `attributes(f)` were naively added to the modified `f`:

```
# Create a function with no dots:
foo <- function(a = 1) {
  # Helpful comment
  a
}

# Give it important attributes that we can't afford to lose:
attr(foo, "important_attribute") <- "crucial information"
class(foo) <- "very_special_function"

# Print foo, which also prints its important attributes:
foo
#> function(a = 1) {
#>   # Helpful comment
#>   a
#> }
#> <environment: 0x571c620>
#> attr(,"important_attribute")
#> [1] "crucial information"
#> attr(,"class")
#> [1] "very_special_function"

# Save its attributes:
old_attributes <- attributes(foo)

# Add dots:
formals(foo)[["..."]] <- quote(expr = )

# See that the important attributes have been dropped:
foo
#> function (a = 1, ...)
#> {
#>   a
#> }
```

```

#> <environment: 0x571c620>

# Add the attributes back:
attributes(foo) <- old_attributes

# Print it again, and we see that the attributes have returned.
# However, the ... disappears from the argument list.
foo
#> function(a = 1) {
#>   # Helpful comment
#>   a
#> }
#> <environment: 0x571c620>
#> attr("important_attribute")
#> [1] "crucial information"
#> attr("class")
#> [1] "very_special_function"

# We know the actual function definitely has dots, since it can handle
# extraneous arguments:
foo(1, 2, junk, "arguments", NULL)
#> [1] 1

# Remove the "srcref" attribute, and the function is printed accurately.
# Furthermore, its important attributes are intact:
attr(foo, "srcref") <- NULL
foo
#> function (a = 1, ...)
#> {
#>   a
#> }
#> <environment: 0x571c620>
#> attr("important_attribute")
#> [1] "crucial information"
#> attr("class")
#> [1] "very_special_function"

# Success (although the comments in the body() of the function are lost)

```

Examples

```

# The base::match() function has no ... and can't handle extraneous arguments
if (FALSE) {
  match("z", letters, cannot_handle_ = "junk arguments")
}

# But if we give it dots...
match_with_dots <- withdots(match)

```

```
# ...it can now handle extraneous arguments:  
match_with_dots("z", letters, can_now_handle = "junk arguments")
```

Index

..., 2, 3

args, 2

attribute, 2, 3

attributes, 2, 3

body, 3

c(), 2

closure, 2

documentation page, 3

formals, 2, 3

function, 2, 3

function(), 3

max(), 2

primitive, 2

print, 3

print.function(), 3

rep(), 2

rlang::as_closure(), 2

srcref, 2, 3

withdots, 2