

# Package ‘sloop’

May 9, 2026

**Title** Helpers for 'OOP' in R

**Version** 1.0.1

**Description** A collection of helper functions designed to help you to better understand object oriented programming in R, particularly using 'S3'.

**License** GPL-3

**URL** <https://github.com/r-lib/sloop>, <https://sloop.r-lib.org>

**BugReports** <https://github.com/r-lib/sloop/issues>

**Depends** R (>= 3.3)

**Imports** codetools, crayon, methods, purrr, rlang, tibble (>= 2.0.1)

**Suggests** covr, testthat

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.1.1

**NeedsCompilation** no

**Author** Hadley Wickham [aut, cre],  
RStudio [cph]

**Maintainer** Hadley Wickham <hadley@rstudio.com>

**Repository** CRAN

**Date/Publication** 2019-02-17 15:30:03 UTC

## Contents

ftype . . . . .	2
is_s3_generic . . . . .	2
otype . . . . .	3
s3_class . . . . .	4
s3_dispatch . . . . .	4
s3_get_method . . . . .	5
s3_methods_class . . . . .	6
<b>Index</b>	<b>7</b>

---

ftype	<i>Determine function type.</i>
-------	---------------------------------

---

**Description**

This function figures out whether the input function is a regular/primitive/internal function, a internal/S3/S4 generic, or a S3/S4/RC method. This is function is slightly simplified as it's possible for a method from one class to be a generic for another class, but that seems like such a bad idea that hopefully no one has done it.

**Usage**

```
ftype(f)
```

**Arguments**

f	unquoted function name
---	------------------------

**Value**

a character of vector of length 1 or 2.

**Examples**

```
ftype(`%in%`)
ftype(sum)
ftype(t.data.frame)
ftype(t.test) # Tricky!
ftype(writelnLines)
ftype(unlist)
```

---

is_s3_generic	<i>Determine if a function is an S3 generic or S3 method.</i>
---------------	---

---

**Description**

is\_s3\_generic() compares name checks for both internal and regular generics. is\_s3\_method() builds names of all possible generics for that function and then checks if any of them actually is a generic.

**Usage**

```
is_s3_generic(fname, env = parent.frame())
```

```
is_s3_method(fname, env = parent.frame())
```

**Arguments**

fname	Name of function as a string. Need name of function because it's impossible to determine whether or not a function is a S3 method based only on its contents.
env	Environment to search in.

**Examples**

```
is_s3_generic("mean")
is_s3_generic("sum")
is_s3_generic("[[")
is_s3_generic("unlist")
is_s3_generic("runif")

is_s3_method("t.data.frame")
is_s3_method("t.test") # Just tricking!
is_s3_method("as.data.frame")
is_s3_method("mean.Date")
```

---

otype

*Determine the type of an object*

---

**Description**

Tells you if you're dealing with an base, S3, S4, RC, or R6 object.

**Usage**

```
otype(x)
```

**Arguments**

x	An object
---	-----------

**Examples**

```
otype(1:10)
otype(mtcars)
```

---

`s3_class`*Compute the S3 class of an object*

---

**Description**

Compared to `class()`, this always returns the class vector that is used for dispatch. This is most important for objects where the class attribute has not been set.

**Usage**

```
s3_class(x)
```

**Arguments**

`x`                    A primitive type

**Examples**

```
s3_class(NULL)

s3_class(logical())
s3_class(integer())
s3_class(numeric())
s3_class(character())

s3_class(matrix())
s3_class(matrix(1))

s3_class(array())
s3_class(array(1))
```

---

`s3_dispatch`*Illustrate S3 dispatch*

---

**Description**

`s3_dispatch()` prints a list of all possible function names that will be considered for method dispatch. There are four possible states:

- `=>` method exists and is found by `UseMethod()`.
- `->` method exists and is used by `NextMethod()`.
- `*` method exists but is not used.
- Nothing (and greyed out in console): method does not exist.

Learn more at <https://adv-r.hadley.nz/s3.html>.

**Usage**

```
s3_dispatch(call, env = parent.frame())
```

**Arguments**

call	Example call to S3 method
env	Environment in which to evaluate call

**Examples**

```
x <- Sys.time()
s3_dispatch(print(x))
s3_dispatch(is.numeric(x))
s3_dispatch(as.Date(x))
s3_dispatch(sum(x))

# Internal vs. regular generic
x1 <- 1
x2 <- structure(2, class = "double")

my_length <- function(x) UseMethod("my_length")
s3_dispatch(my_length(x1))
s3_dispatch(my_length(x2))

length.double <- function(x) 10
s3_dispatch(length(x1))
s3_dispatch(length(x2))
```

---

s3_get_method	<i>Find S3 method from its name</i>
---------------	-------------------------------------

---

**Description**

Find S3 method from its name

**Usage**

```
s3_get_method(name)
```

**Arguments**

name	A string or unquoted symbol
------	-----------------------------

**Value**

A function, or an error stating why the method could not be found

## Examples

```
s3_get_method(mean.Date)
s3_get_method(weighted.mean.Date)
```

---

s3_methods_class	<i>List methods for a S3 or S4 generic (or class)</i>
------------------	---

---

## Description

Returns information about all methods belong to a generic or a class. In S3 and S4, methods belong to a generic, but it is often useful to see what generics have been provided methods for a given class. These are wrappers around `utils::methods()`, which returns a lot of useful information in an attribute.

## Usage

```
s3_methods_class(x)
s3_methods_generic(x)
s4_methods_class(x)
s4_methods_generic(x)
```

## Arguments

x	Name of class or generic
---	--------------------------

## Value

A tibble with columns `generic`, `visible`, `class`, `visible`, and `source`.

## Examples

```
s3_methods_class("Date")
s3_methods_generic("anova")

s4_methods_class("Date")
s4_methods_generic("anova")
```

# Index

## \* object inspection

ftype, 2

class(), 4

ftype, 2

is\_s3\_generic, 2

is\_s3\_method(is\_s3\_generic), 2

otype, 3

s3\_class, 4

s3\_dispatch, 4

s3\_get\_method, 5

s3\_methods\_class, 6

s3\_methods\_generic(s3\_methods\_class), 6

s4\_methods\_class(s3\_methods\_class), 6

s4\_methods\_generic(s3\_methods\_class), 6

utils::methods(), 6