

# Package ‘evaluate’

February 18, 2022

**Type** Package

**Title** Parsing and Evaluation Tools that Provide More Details than the Default

**Version** 0.15

**Description** Parsing and evaluation tools that make it easy to recreate the command line behaviour of R.

**License** MIT + file LICENSE

**URL** <https://github.com/r-lib/evaluate>

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

**Depends** R (>= 3.0.2)

**Imports** methods

**Suggests** covr, ggplot2, lattice, testthat

**RoxygenNote** 7.1.1

**Encoding** UTF-8

**NeedsCompilation** no

**Author** Hadley Wickham [aut],

Yihui Xie [aut, cre] (<<https://orcid.org/0000-0003-0645-5666>>),

Michael Lawrence [ctb],

Thomas Kluyver [ctb],

Jeroen Ooms [ctb],

Barret Schloerke [ctb],

Adam Ryzkowski [ctb],

Hiroaki Yutani [ctb],

Michel Lang [ctb],

Karolis Koncevičius [ctb]

**Maintainer** Yihui Xie <[xie@yihui.name](mailto:xie@yihui.name)>

**Repository** CRAN

**Date/Publication** 2022-02-18 17:20:02 UTC

## R topics documented:

evaluate . . . . .	2
flush_console . . . . .	3
new_output_handler . . . . .	4
parse_all . . . . .	5
replay . . . . .	5
<b>Index</b>	<b>7</b>

---

evaluate	<i>Evaluate input and return all details of evaluation.</i>
----------	-------------------------------------------------------------

---

### Description

Compare to `eval()`, `evaluate` captures all of the information necessary to recreate the output as if you had copied and pasted the code into a R terminal. It captures messages, warnings, errors and output, all correctly interleaved in the order in which they occurred. It stores the final result, whether or not it should be visible, and the contents of the current graphics device.

### Usage

```
evaluate(
  input,
  envir = parent.frame(),
  enclos = NULL,
  debug = FALSE,
  stop_on_error = 0L,
  keep_warning = TRUE,
  keep_message = TRUE,
  new_device = TRUE,
  output_handler = default_output_handler,
  filename = NULL,
  include_timing = FALSE
)
```

### Arguments

input	input object to be parsed and evaluated. May be a string, file connection or function. Passed on to <code>parse_all()</code> .
envir	environment in which to evaluate expressions.
enclos	when <code>envir</code> is a list or data frame, this is treated as the parent environment to <code>envir</code> .
debug	if TRUE, displays information useful for debugging, including all output that <code>evaluate</code> captures.

stop_on_error	if 2, evaluation will halt on first error and you will get no results back. If 1, evaluation will stop on first error without signaling the error, and you will get back all results up to that point. If 0 will continue running all code, just as if you'd pasted the code into the command line.
keep_warning, keep_message	whether to record warnings and messages.
new_device	if TRUE, will open a new graphics device and automatically close it after completion. This prevents evaluation from interfering with your existing graphics environment.
output_handler	an instance of <code>output_handler()</code> that processes the output from the evaluation. The default simply prints the visible return values.
filename	string overriding the <code>base::srcfile()</code> filename.
include_timing	if TRUE, evaluate will wrap each input expression in <code>system.time()</code> , which will be accessed by following <code>replay()</code> call to produce timing information for each evaluated command.

---

flush\_console

*An emulation of `flush.console()` in `evaluate()`*

---

## Description

When `evaluate()` is evaluating code, the text output is diverted into an internal connection, and there is no way to flush that connection. This function provides a way to "flush" the connection so that any text output can be immediately written out, and more importantly, the text handler (specified in the `output_handler` argument of `evaluate()`) will be called, which makes it possible for users to know it when the code produces text output using the handler.

## Usage

```
flush_console()
```

## Note

This function is supposed to be called inside `evaluate()` (e.g. either a direct `evaluate()` call or in **knitr** code chunks).

---

new\_output\_handler      *Custom output handlers.*

---

### Description

An output\_handler handles the results of `evaluate()`, including the values, graphics, conditions. Each type of output is handled by a particular function in the handler object.

### Usage

```
new_output_handler(  
  source = identity,  
  text = identity,  
  graphics = identity,  
  message = identity,  
  warning = identity,  
  error = identity,  
  value = render,  
  calling_handlers = list()  
)
```

### Arguments

source	Function to handle the echoed source code under evaluation.
text	Function to handle any textual console output.
graphics	Function to handle graphics, as returned by <code>recordPlot()</code> .
message	Function to handle <code>message()</code> output.
warning	Function to handle <code>warning()</code> output.
error	Function to handle <code>stop()</code> output.
value	Function to handle the values returned from evaluation. If it only has one argument, only visible values are handled; if it has more arguments, the second argument indicates whether the value is visible.
calling_handlers	List of <a href="#">calling handlers</a> . These handlers have precedence over the exiting handler installed by <code>evaluate()</code> when <code>stop_on_error</code> is set to 0.

### Details

The handler functions should accept an output object as their first argument. The return value of the handlers is ignored, except in the case of the value handler, where a visible return value is saved in the output list.

Calling the constructor with no arguments results in the default handler, which mimics the behavior of the console by printing visible values.

Note that recursion is common: for example, if value does any printing, then the text or graphics handlers may be called.

**Value**

A new output\_handler object

---

parse_all	<i>Parse, retaining comments.</i>
-----------	-----------------------------------

---

**Description**

Works very similarly to parse, but also keeps original formatting and comments.

**Usage**

```
parse_all(x, filename = NULL, allow_error = FALSE)
```

**Arguments**

x	object to parse. Can be a string, a file connection, or a function. If a connection, will be opened and closed only if it was closed initially.
filename	string overriding the file name
allow_error	whether to allow syntax errors in x

**Value**

A data.frame with columns src, the source code, and expr. If there are syntax errors in x and allow\_error = TRUE, the data frame has an attribute PARSE\_ERROR that stores the error object.

---

replay	<i>Replay a list of evaluated results.</i>
--------	--------------------------------------------

---

**Description**

Replay a list of evaluated results, as if you'd run them in an R terminal.

**Usage**

```
replay(x)
```

**Arguments**

x	result from <code>evaluate()</code>
---	-------------------------------------

**Examples**

```
samples <- system.file("tests", "testthat", package = "evaluate")
if (file_test("-d", samples)) {
  replay(evaluate(file(file.path(samples, "order.r"))))
  replay(evaluate(file(file.path(samples, "plot.r"))))
  replay(evaluate(file(file.path(samples, "data.r"))))
}
```

# Index

`base::srcfile()`, 3  
calling handlers, 4  
`eval()`, 2  
evaluate, 2  
`evaluate()`, 3–5  
`flush_console`, 3  
`message()`, 4  
`new_output_handler`, 4  
`output_handler (new_output_handler)`, 4  
`output_handler()`, 3  
`parse_all`, 5  
`parse_all()`, 2  
`recordPlot()`, 4  
replay, 5  
`stop()`, 4  
`warning()`, 4