Package: evaluate (via r-universe)

July 3, 2024

Type Package

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

Version 0.24.0.9000

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

License MIT + file LICENSE

URL `https://evaluate.r-lib.org/`, `https://github.com/r-lib/evaluate`

BugReports `https://github.com/r-lib/evaluate/issues`

Depends R (>= 3.6.0)

Suggests covr, ggplot2 (>= 3.3.6), lattice, methods, rlang, testthat (>= 3.0.0), withr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Repository https://r-lib.r-universe.dev

RemoteUrl https://github.com/r-lib/evaluate

RemoteRef HEAD

RemoteSha e00724f6a39039b0d1c7c5fec3a74b3d473a6a53

Contents

evaluate *Evaluate input and return all details of evaluation*

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 occured. 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,
  log_echo = FALSE,
  log_warning = FALSE,
  new_device = TRUE,
  output_handler = NULL,
  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 `parse_all()`.
`envir`	environment in which to evaluate expressions.
`enclos`	when `envir` is a list or data frame, this is treated as the parent environment to `envir`.
`debug`	if `TRUE`, displays information useful for debugging, including all output that evaluate captures.
`stop_on_error`	A number between 0 and 2 that controls what happens when the code errors: • If `0`, the default, will continue running all code, just as if you'd pasted the code into the command line. • If `1`, evaluation will stop on first error without signaling the error, and you will get back all results up to that point. • If `2`, evaluation will halt on first error and you will get back no results.

`keep_warning`, `keep_message`
: A single logical value that controls what happens to warnings and messages.

- If `TRUE`, the default, warnings and messages will be captured in the output.
- If `NA`, warnings and messages will not be captured and bubble up to the calling environment of `evaluate()`.
- If `FALSE`, warnings and messages will be completed supressed and not shown anywhere.

Note that setting the envvar `R_EVALUATE_BYPASS_MESSAGES` to `true` will force these arguments to be set to `NA`.

`log_echo`, `log_warning`
: If `TRUE`, will immediately log code and warnings (respectively) to `stderr`.

This will be force to `TRUE` if env var `ACTIONS_STEP_DEBUG` is `true`, as when debugging a failing GitHub Actions workflow.

`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 `output_handler()` that processes the output from the evaluation. The default simply prints the visible return values.

`filename`
: string overrriding the `base::srcfile()` filename.

`include_timing`
: Deprecated.

Examples

```
evaluate(c(
  "1 + 1",
  "2 + 2"
))

# Not that's there's a difference in output between putting multiple
# expressions on one line vs spreading them across multiple lines
evaluate("1;2;3")
evaluate(c("1", "2", "3"))

# This also affects how errors propagate, matching the behaviour
# of the R console
evaluate("1;stop(2);3")
evaluate(c("1", "stop(2)", "3"))
```

`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.

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

Usage

```
flush_console()
```

---

local_reproducible_output

Control common output options

Description

Often when using `evaluate()` you are running R code with a specific output context in mind. But there are many options and env vars that packages will take from the current environment, meaning that output depends on the current state in undesirable ways.

This function allows you to describe the characteristics of the desired output and takes care of setting the options and environment variables for you.

Usage

```
local_reproducible_output(
  width = 80,
  color = FALSE,
  unicode = FALSE,
  hyperlinks = FALSE,
  rstudio = FALSE,
  frame = parent.frame()
)
```

Arguments

`width`	Value of the `"width"` option.
`color`	Determines whether or not cli/crayon colour should be used.
`unicode`	Should we use unicode characaters where possible?
`hyperlinks`	Should we use ANSI hyperlinks?
`rstudio`	Should we pretend that we're running inside of RStudio?
`frame`	Scope of the changes; when this calling frame terminates the changes will be undone. For expert use only.

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. This function should take two arguments (src and expr), and return an object that will be inserted into the evaluate outputs. src is the unparsed text of the source code, and expr is the complete input expression (which may have 0, 1, 2, or more components; see parse_all() for details). Return src for the default evaluate behaviour. Return NULL to drop the source from the output.
text	Function to handle any textual console output.
graphics	Function to handle graphics, as returned by recordPlot().
message	Function to handle message() output.
warning	Function to handle warning() output.
error	Function to handle stop() output.
value	Function to handle the values returned from evaluation. • If it has one argument, it called on visible values. • If it has two arguments, it handles all values, with the second argument indicating whether or not the value is visible.
calling_handlers	List of calling handlers. These handlers have precedence over the exiting handler installed by evaluate() when stop_on_error 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` *Parse, retaining comments*

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 two columns, `src` and `expr`, and one row for each complete input in `x`. A complete input is R code that would trigger execution when typed at the console. This might consist of multiple expressions separated by `;` or one expression spread over multiple lines (like a function definition).

`src` is a character vector of source code. Each element represents a complete input expression (which might span multiple line) and always has a terminal `\n`.

`expr` is a list-column of expressions. The expressions can be of any length, depending on the structure of the complete input source:

- If `src` consists of only only whitespace and/or comments, `expr` will be length 0.
- If `src` a single scalar (like `TRUE`, `1`, or `"x"`), name, or function call, `expr` will be length 1.
- If `src` contains multiple expressions separated by `;`, `expr` will have length two or more.

The expressions have their srcrefs removed.

If there are syntax errors in `x` and `allow_error = TRUE`, the data frame will have an attribute `PARSE_ERROR` that stores the error object.

Examples

```
# Each of these inputs are single line, but generate different numbers of
# expressions
source <- c(
  "# a comment",
  "x",
  "x;y",
  "x;y;z"
)
parsed <- parse_all(source)
lengths(parsed$expr)
str(parsed$expr)

# Each of these inputs are a single expression, but span different numbers
# of lines
source <- c(
  "function() {}",
  "function() {",
  "  # Hello!",
  "}",
  "function() {",
  "  # Hello!",
  "  # Goodbye!",
  "}"
)
parsed <- parse_all(source)
lengths(parsed$expr)
parsed$src
```

`replay` *Replay a list of evaluated results*

Description

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

Usage

```
replay(x)
```

Arguments

`x`	result from `evaluate()`

Examples

```
f1 <- function() {
  cat("1\n")
  print("2")
  warning("3")
  print("4")
  message("5")
  stop("6")
}
replay(evaluate("f1()"))

f2 <- function() {
  message("Hello")
  plot(1:10)
  message("Goodbye")
}
replay(evaluate("f2()"))
```

Index