

# Package ‘epl’y

May 8, 2026

**Type** Package

**Title** Apply a Function Over Expressions

**Version** 0.1.2

**Description** Evaluate a function over a data frame of expressions.

**License** GPL-3

**Depends** R (>= 3.0.0)

**Imports** magrittr, methods

**Suggests** testthat, knitr, rmarkdown

**VignetteBuilder** knitr

**URL** <https://github.com/wlandau/epl'y>

**BugReports** <https://github.com/wlandau/epl'y/issues>

**RoxygenNote** 6.0.1

**NeedsCompilation** no

**Author** William Michael Landau [aut, cre],  
Eli Lilly and Company [cph]

**Maintainer** William Michael Landau <will.landau@gmail.com>

**Repository** CRAN

**Date/Publication** 2018-04-06 16:35:43 UTC

## Contents

epl'y-package	2
epl'y	2
evals	3
example.expr	4
example.fun	5
example.with	5
help_epl'y	6
quotes	6
strings	7
unquote	8

**Index****9**


---

epl-package	<i>The epl package provides ways to call <code>eval(parse(text = ...))</code> in bulk. The <code>evals()</code> function is a vectorized version of <code>eval(parse(text = ...))</code>. <code>epl()</code> is like <code>apply(MARGIN = 1)</code> except that the elements of each row are <code>eval(parse(text = ...))</code>'ed before being supplied to the function.</i>
-------------	---

---

**Description**

The epl package provides ways to call `eval(parse(text = ...))` in bulk. The `evals()` function is a vectorized version of `eval(parse(text = ...))`. `epl()` is like `apply(MARGIN = 1)` except that the elements of each row are `eval(parse(text = ...))`'ed before being supplied to the function.

**Author(s)**

William Michael Landau <will.landau@gmail.com>

**References**

<<https://github.com/wlandau/epl>>

**Examples**

```
# Get an example data frame of commands that evaluate to function arguments.
.expr <- example.expr()
.fun <- example.fun # Get an example collection of functions.
# Get an example list of supporting data. Could be an environment.
.with <- example.with()
# Row-by-row, evaluate the code in .expr and feed the results to the function.
epl(.fun = .fun, .expr = .expr, .with = .with)
evals(x = c("a + 1", "b + 2"), .with = .with)
```

---

epl

*Function epl*


---

**Description**

Apply a function over a data frame of quoted expressions. Parallel execution is available using the `.split` and `.tasks` arguments.

**Usage**

```
epl(.fun, .expr, .with = parent.frame())
```

**Arguments**

<code>.fun</code>	function to evaluate.
<code>.expr</code>	data frame of quoted expressions. Column names must contain the argument names of <code>.fun</code> .
<code>.with</code>	list, data frame, or environment with the data accessible to <code>.expr</code>

**Details**

`.fun` is a function, and `.expr` is a data frame. In `.expr`, each row stands for a single call to `.fun`, and each column stands for an argument. Each element is a quoted expression that uses the data in `.with` during evaluation. When `evals` is called on each row, the expressions are evaluated on `.with`, and the results are given to `.fun` as function arguments. The column names of `.expr` must contain the argument names of `.fun`. With `.tasks` and `.split`, Mac and Linux users can distribute the work over multiple parallel tasks. See the vignette for an example (`vignette("evals")`).

**Value**

a list or vector of return values of `.fun`.

**See Also**

[evals](#), [help\\_evals](#)

**Examples**

```
# Get an example data frame of commands that evaluate to function arguments.
.expr <- example.expr()
.fun <- example.fun # Get an example collection of functions.
# Get an example list of supporting data. Could be an environment.
.with <- example.with()
# Row-by-row, evaluate the code in .expr and feed the results to the function.
evals(.fun = .fun, .expr = .expr, .with = .with)
```

---

evals

*Function evals*


---

**Description**

Evaluate a character vector as a bunch of expressions.

**Usage**

```
evals(x = NULL, .with = parent.frame(), .simplify = TRUE)
```

**Arguments**

<code>x</code>	character vector of expressions to evaluate
<code>.with</code>	list, data frame, or environment with the data accessible to the expressions in <code>x</code>
<code>.simplify</code>	TRUE to simplify the result and FALSE otherwise

**Value**

a list or vector of return values of `.fun`.

**See Also**

[epl](#), [help\\_epl](#)

**Examples**

```
# Get an example list of supporting data. Could be an environment.
.with <- example.with()
# Row-by-row, evaluate the code in .expr and feed the results to the function.
evals(x = c("a + 1", "b + 2"), .with = .with)
```

---

example.expr

example.expr

---

**Description**

Return example `.expr` argument for [epl](#).

**Usage**

```
example.expr()
```

**Value**

Example `.expr` argument to [epl](#).

**See Also**

[epl](#)

**Examples**

```
## Get an example .expr argument to epl().
## See the examples of the epl() function for usage.
example.expr()
```

---

example.fun	example.fun
-------------	-------------

---

**Description**

Example .fun argument to [epley](#).

**Usage**

```
example.fun(x, y)
```

**Arguments**

x	numeric argument
y	nonzero numeric argument

**Value**

Example .fun argument to [epley](#).

**See Also**

[epley](#)

**Examples**

```
## Get an example .fun argument to epley().  
## See the examples of the epley() function for usage.  
example.fun  
example.fun(x = c(4, 2), y = c(2, 2))
```

---

example.with	example.with
--------------	--------------

---

**Description**

Return example .with argument of [epley](#).

**Usage**

```
example.with()
```

**Value**

example .with argument of [epley](#)

**See Also**[eplly](#)**Examples**

```
#' Get an example .with argument to eply() and evals().
#' See the examples of the eply() and evals() functions for usage.
example.with()
```

---

help_eply	<i>Function</i> help_eply
-----------	---------------------------

---

**Description**

Prints links for tutorials, troubleshooting, bug reports, etc.

**Usage**

```
help_eply()
```

**See Also**[eplly](#), [evals](#)**Examples**

```
help_eply()
```

---

quotes	<i>Function</i> quotes
--------	------------------------

---

**Description**

Put quotes around each element of a character vector.

**Usage**

```
quotes(x = NULL, single = FALSE)
```

**Arguments**

<code>x</code>	character vector or object to be coerced to character.
<code>single</code>	Add single quotes if TRUE and double quotes otherwise.

**Value**

character vector with quotes around it

**See Also**

[unquote](#), [strings](#), [epl](#), [help\\_epl](#)

**Examples**

```
quotes(letters[1:3])  
quotes(letters[1:3], single = TRUE)  
quotes(letters[1:3], single = FALSE)
```

---

strings	<i>Function</i> strings
---------	-------------------------

---

**Description**

Turn valid expressions into character strings.

**Usage**

```
strings(...)
```

**Arguments**

... unquoted symbols to turn into character strings.

**Value**

a character vector

**See Also**

[quotes](#), [unquote](#), [epl](#), [help\\_epl](#)

**Examples**

```
strings(a, b, bee)
```

---

unquote	<i>Function</i> unquote
---------	-------------------------

---

**Description**

Remove leading and trailing escaped quotes from character strings.

**Usage**

```
unquote(x = NULL, deep = FALSE)
```

**Arguments**

x	character vector
deep	remove all outer quotes if TRUE and only the outermost set otherwise. Single and double quotes are treated interchangeably, and matching is not checked.

**Value**

character vector without leading or trailing escaped quotes around it

**See Also**

[quotes](#), [strings](#), [reply](#), [help\\_reply](#)

**Examples**

```
unquote(c("x", "'y'", "\"why\"", "'z'"))
unquote(c("x", "'y'", "\"why\"", "'z'"), deep = FALSE)
unquote(c("x", "'y'", "\"why\"", "'z'"), deep = TRUE)
```

# Index

`reply`, [2](#), [2](#), [3–8](#)  
`reply-package`, [2](#)  
`evals`, [2](#), [3](#), [3](#), [6](#)  
`example.expr`, [4](#)  
`example.fun`, [5](#)  
`example.with`, [5](#)

`help_reply`, [3](#), [4](#), [6](#), [7](#), [8](#)

`quotes`, [6](#), [7](#), [8](#)

`strings`, [7](#), [7](#), [8](#)

`unquote`, [7](#), [8](#)