

# Package: pkgcond (via r-universe)

October 13, 2024

**Type** Package

**Title** Classed Error and Warning Conditions

**Version** 0.1.1

**Maintainer** Andrew Redd <Andrew.Redd@hsc.utah.edu>

**Description** This provides utilities for creating classed error and warning conditions based on where the error originated.

**License** GPL-2

**Encoding** UTF-8

**Depends** R(>= 3.5.0)

**Imports** assertthat, methods

**Suggests** covr, testthat

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.1.1

**Language** en-US

**Collate** 'assert\_that.R' 'conditions.R' 'comma\_list.R' 'find\_scope.R' 'infix.R' 'skip\_scope.R' 'suppress.R' 'translate.R'

**URL** <https://github.com/RDocTaskForce/pkgcond>

**BugReports** <https://github.com/RDocTaskForce/pkgcond/issues>

**Repository** <https://rdoctaskforce.r-universe.dev>

**RemoteUrl** <https://github.com/rdoctaskforce/pkgcond>

**RemoteRef** HEAD

**RemoteSha** f57f1c3b5832b4912a39e644a81881d548c41e70

## Contents

|                       |   |
|-----------------------|---|
| assert_that . . . . . | 2 |
| collapse . . . . .    | 3 |
| comma_list . . . . .  | 3 |
| condition . . . . .   | 4 |

|                               |   |
|-------------------------------|---|
| dot-underscore . . . . .      | 5 |
| find_scope . . . . .          | 5 |
| infix-concatenation . . . . . | 6 |
| not-in . . . . .              | 7 |
| skip_scope . . . . .          | 7 |
| suppress . . . . .            | 8 |

|              |           |
|--------------|-----------|
| <b>Index</b> | <b>10</b> |
|--------------|-----------|

---

|             |                          |
|-------------|--------------------------|
| assert_that | <i>Scoped Assertions</i> |
|-------------|--------------------------|

---

## Description

The pkgcond package intentionally overrides the `assertthat::assert_that()` function. It provides the same utility but enhances the original version by throwing scoped and typed errors. The type is 'assertion failure' and the scope can be set or inferred from the calling frame.

## Usage

```
assert_that(
  ...,
  env = parent.frame(),
  msg = NULL,
  scope = find_scope(env),
  type = "assertion failure"
)
```

## Arguments

|       |   |
|-------|---|
| ...   | unnamed expressions that describe the conditions to be tested. Rather than combining expressions with <code>&amp;&amp;</code> , separate them by commas so that better error messages can be generated. |
| env   | (advanced use only) the environment in which to evaluate the assertions.  |
| msg   | a custom error message to be printed if one of the conditions is false.   |
| scope | The scope of the error.   |
| type  | The error type.   |

---

|          |                                   |
|----------|-----------------------------------|
| collapse | <i>Collapse character Vectors</i> |
|----------|-----------------------------------|

---

**Description**

Collapse character Vectors

**Usage**

```
collapse(x, with = " ")
```

```
collapse0(x, with = "")
```

**Arguments**

|      |   |
|------|---|
| x    | a character vector                        |
| with | character to place between elements of x. |

---

|            |   |
|------------|---|
| comma_list | <i>Construct a comma separated list</i> |
|------------|---|

---

**Description**

Use this utility to create nicely formatted lists for error messages and the like.

**Usage**

```
comma_list(x, sep = ", ", sep2 = " and ", sep.last = ", and ", terminator = "")
```

**Arguments**

|            |  |
|------------|--|
| x          | a list that can be converted into a character.   |
| sep        | the typical separator  |
| sep2       | the separator to use in the case of only two elements.   |
| sep.last   | the separator to use between the last and next to last elements when there are at least 3 element in the list. |
| terminator | concatenated to the end after the list is concluded.   |

**Examples**

```
comma_list(c("you", "I"))
comma_list(c("you", "I"), sep2=" & ")
comma_list(head(letters), sep.last=', ', term=', ...')
```

---

condition

*Raise a mutable and classed condition.*


---

### Description

Raising Classed conditions helps with catching errors. These allow for typing errors as they arise and adding scopes to better catch errors from specific locations.

### Usage

```
condition(
  msg,
  cond = .conditions,
  ...,
  scope = find_scope(),
  type = NULL,
  call = sys.call(1)
)
```

```
pkg_error(msg, ..., scope = find_scope(), call = sys.call(1))
```

```
pkg_warning(msg, ..., scope = find_scope(), call = sys.call(1))
```

```
pkg_message(msg, ..., scope = find_scope(), call = sys.call(1))
```

### Arguments

|       |  |
|-------|--|
| msg   | The message to convey  |
| cond  | The severity of the condition, or what to do; give a 'message' (default), a 'warning', an 'error' or do 'none' and ignore.   |
| ...   | Attributes to be added to condition object for condition, arguments passed to condition for all others.  |
| scope | A character vector of the scope(s) of the signal. Defaults to the package name but could be longer such as package name, a class name, and a method call. This should be used as a where the error occurred. |
| type  | Used with scope and cond to set the class of the condition object to raise. This should be a type of error; out of bounds, type mismatch, etcetera.  |
| call  | The call to use to include in the condition.   |

### Details

The `condition()` function alone provides a flexible and dynamic way of producing conditions in code. The functions `pkg_error`, `pkg_warning`, and `pkg_message` do the same as `condition` except restricted to errors, warnings, and messages respectively.

---

dot-underscore      *Format and Translate Strings*

---

### Description

This shortcut provides simple translation and formatting functionality. Essentially it is a wrapper for `base::gettext()` and `base::gettextf()`.

### Usage

```
._(msg, ..., domain = NULL)
```

### Arguments

|        |   |
|--------|---|
| msg    | The message to translate.                                       |
| ...    | Arguments passed on to <code>base::gettextf</code>              |
| fmt    | a character vector of format strings, each of up to 8192 bytes. |
| domain | see <code>base::gettext()</code>                                |

### Examples

```
loki <- list()
class(loki) <- "puny god"
._("I am a %s.", class(loki))
```

---

find\_scope      *Find the default scope of a call.*

---

### Description

This find the scope of the call. It includes the package of the call, the class if called from a method, and the name of the function called.

### Usage

```
find_scope(frame = NULL, global = FALSE)
```

### Arguments

|        |   |
|--------|---|
| frame  | The frame to infer scope from.                  |
| global | Should the global frame be listed in the scope. |

## Examples

```
my_function <- function(){
  scope <- find_scope()
  "You are in" %<<% collapse(scope, '::')
}
my_function()

my_sights <- my_function
my_sights()
```

---

infix-concatenation    *Infix string concatenation.*

---

## Description

The infix operators listed here are three versions of paste.

- %\% is for preserving line breaks
- %<<% is an infix replacement for [paste](#)
- %<<<% is paste with no space and no break."

## Usage

```
lhs %<<% rhs
```

```
lhs %<<<% rhs
```

## Arguments

```
lhs            left string
```

```
rhs            right string
```

## Examples

```
who <- "world"
'hello_' %<<<% who

'Sing with me' %<<% head(letters) %<<% '...'
```

---

|        |                              |
|--------|------------------------------|
| not-in | <i>Not in infix operator</i> |
|--------|------------------------------|

---

**Description**

The same as `%in%` but negated.

**Usage**

```
x %!in% table
```

**Arguments**

|       |   |
|-------|---|
| x     | vector or NULL: the values to be matched. <a href="#">Long vectors</a> are supported.             |
| table | vector or NULL: the values to be matched against. <a href="#">Long vectors</a> are not supported. |

**Examples**

```
'A' %!in% letters #TRUE letters are lower case.  
'A' %!in% LETTERS #FALSE LETTERS are upper case.
```

---

|            |   |
|------------|---|
| skip_scope | <i>Exclude a function from find_scope</i> |
|------------|---|

---

**Description**

In the course of work it will often be the case that one would like to create a new condition function, such such as for specific errors or warning. These should not be included in the scope when inferred. The natural solution would be to include the scope in every call to condition or have it inferred in each function definition. This however, gets very tedious.

**Usage**

```
skip_scope(fun)
```

**Arguments**

|     |                   |
|-----|-------------------|
| fun | a function to tag |
|-----|-------------------|

**Details**

The `skip_scope` function tags a function as one that should be excluded from consideration when determining scope via `find_scope()`.

**Value**

The fun function with the skipscope attribute set to TRUE.

**Examples**

```
new_msg <- function(when=find_scope()){
  "Hello from" %<<% when
}
new_postcard <- function(msg){
  greeting <- new_msg()
  paste0(greeting, '\n\n', msg)
}

cat(new_postcard("Not all is well"), '\n')
new_msg <- skip_scope(new_msg)

cat(new_postcard("Now all is well"))
```

---

suppress

*Selectively suppress warnings and messages*

---

**Description**

This collection of functions allow the suppression of condition messages, warnings and messages, through filtering the condition message, the condition class or a combination of the two.

**Usage**

```
suppress_conditions(expr, pattern = NULL, class = NULL, ...)
suppress_warnings(expr, pattern = NULL, class = "warning", ...)
suppress_messages(expr, pattern = NULL, class = "message", ...)
```

**Arguments**

|         |   |
|---------|---|
| expr    | An expression to evaluate.  |
| pattern | A regular expression pattern to match on.   |
| class   | The class or classes that you would like to filter. When more than one is given the condition may match any of the classes. |
| ...     | Arguments passed on to <code>base::grep</code>  |

x a character vector where matches are sought, or an object which can be coerced by `as.character` to a character vector. **Long vectors** are supported. `ignore.case` if FALSE, the pattern matching is *case sensitive* and if TRUE, case is ignored during matching.

`perl` logical. Should Perl-compatible regexps be used?



fixed logical. If TRUE, pattern is a string to be matched as is. Overrides all conflicting arguments.

useBytes logical. If TRUE the matching is done byte-by-byte rather than character-by-character. See 'Details'.

## Functions

- `suppress_conditions`: The general case of suppressing both messages and warnings.
- `suppress_warnings`: A convenience wrapper that specifies warning class to suppress.
- `suppress_messages`: A convenience wrapper that specifies warning class to suppress.

## Examples

```
## Not run:
testit <- function(){
  warning("this function does nothing.")
  warning("it's pretty useless.")
}
suppress_warning(testit(), "useless") # Will suppress only the second warning by pattern
```

```
# If my_pkg used pkgcond for conditions,
# This would suppress all messages and warnings originating
# in my_pkg functions.
suppress_conditions(my_function(), class='my_pkg-condition')
```

```
## End(Not run)
```

# Index

`._` (dot-underscore), 5  
`%!in%` (not-in), 7  
`%<<%` (infix-concatenation), 6  
`%<%` (infix-concatenation), 6  
`%%` (infix-concatenation), 6  
`%in%`, 7

`assert_that`, 2  
`assertthat::assert_that()`, 2

`base::gettext()`, 5  
`base::gettextf`, 5  
`base::gettextf()`, 5  
`base::grepl`, 8

`collapse`, 3  
`collapse0` (`collapse`), 3  
`comma_list`, 3  
`condition`, 4

`dot-underscore`, 5

`find_scope`, 5  
`find_scope()`, 7

`infix-concatenation`, 6

Long vectors, 7, 8

`not-in`, 7

`paste`, 6  
`pkg_error` (`condition`), 4  
`pkg_message` (`condition`), 4  
`pkg_warning` (`condition`), 4

`skip_scope`, 7  
`suppress`, 8  
`suppress_conditions` (`suppress`), 8  
`suppress_messages` (`suppress`), 8  
`suppress_warnings` (`suppress`), 8