Package ‘assertive.code’

October 21, 2018

Type Package
Title Assertions to Check Properties of Code
Version 0.0-3
Date 2018-10-21
Author Richard Cotton [aut, cre]
Maintainer Richard Cotton <richierocks@gmail.com>
Description A set of predicates and assertions for checking the properties of
code. This is mainly for use by other package developers who want to include
run-time testing features in their own packages. End-users will usually want to
use assertive directly.

URL https://bitbucket.org/richierocks/assertive.code
BugReports https://bitbucket.org/richierocks/assertive.code/issues
Depends R (>= 3.0.0)
Imports assertive.base (>= 0.0-2), assertive.properties,
assertive.types, methods
Suggests testthat
License GPL (>= 3)
LazyLoad yes
LazyData yes
Acknowledgments Development of this package was partially funded by
the Proteomics Core at Weill Cornell Medical College in Qatar
<http://qatar-weill.cornell.edu>. The Core is supported by
‘Biomedical Research Program’ funds, a program funded by Qatar
Foundation.

Collate ‘imports.R’ ‘assert-is-code.R’ ‘is-code.R’

RoxygenNote 6.1.0
NeedsCompilation no
Repository CRAN
Date/Publication 2018-10-21 19:00:02 UTC
assert_all_are_existing

Does the variable exist?

Description

Checks to see if the input variables exist.

Usage

assert_all_are_existing(x, envir = parent.frame(), inherits = TRUE,
                        severity = getOption("assertive.severity", "stop"))

assert_any_are_existing(x, envir = parent.frame(), inherits = TRUE,
                         severity = getOption("assertive.severity", "stop"))

is_existing(x, envir = parent.frame(), inherits = TRUE,
            .xname = get_name_in_parent(x))

Arguments

x  Input to check.
envir  Passed to exists.
inherits  Passed to exists.
severity  How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
.xname  Not intended to be used directly.

Value

is_existing is a vectorized wrapper to exists, providing more information on failure (and with a simplified interface). The assert_* functions return nothing but throw an error if is_existing returns FALSE.
assert_all_are_valid_variable_names

Is the string a valid variable name?

Description

Checks strings to see if they are valid variable names.

Usage

assert_all_are_valid_variable_names(x, allow_reserved = TRUE, allow_duplicates, na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

assert_any_are_valid_variable_names(x, allow_reserved = TRUE, allow_duplicates, na_ignore = FALSE, severity = getOption("assertive.severity", "stop"))

is_valid_variable_name(x, allow_reserved = TRUE, allow_duplicates)

Arguments

x
Input to check.

allow_reserved
If TRUE then "..." and ".1", ".2", etc. are considered valid.

allow_duplicates
Deprecated and ignored.

na_ignore
A logical value. If FALSE, NA values cause an error; otherwise they do not. Like na.rm in many stats package functions, except that the position of the failing values does not change.

severity
How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".

Examples

e <- new.env()
e$x <- 1
e$y <- 2
assert_all_are_existing(c("x", "y"), envir = e)
#These examples should fail.
assertive.base::dont_stop(assert_all_are_existing(c("x", "z"), envir = e))
assert_has_arg

Description
Checks to see if the current call has an argument with the name given in the input.

Usage
assert_has_arg(x, fn = sys.function(sys.parent()), severity = getOption("assertive.severity", "stop"))

has_arg(x, fn = sys.function(sys.parent()))

has_arg_(x, fn = sys.function(sys.parent()))

Arguments
x Argument to check.
fn Function to find the argument in.
severity How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".

Examples

make_random_string <- function(n)
{
  paste0(sample(letters, n, replace = TRUE), collapse = "")
}
long <- c(make_random_string(10000), make_random_string(10001))
x <- c("x", "y_0.Y", ".", "x_y", "...", ".1", long)
nname(is_valid_variable_name(x))
nname(is_valid_variable_name(x, allow_reserved = FALSE))
#These examples should fail.
assertive.base::dont_stop(
  assert_all_are_valid_variable_names(c("...", ".1"), allow_reserved = FALSE)
)

assert_has_arg Does the current call have an argument?

Value
The assert_* functions return nothing but throw an error if the corresponding is_* function returns FALSE.

References
http://4dpiecharts.com/2011/07/04/testing-for-valid-variable-names/

See Also
make.names.
assert_is_binding_locked

Value

has_arg reimplements hasArg, letting you choose the function to search in, and providing more information on failure.

Note

has_arg is for interactive use and takes an unquoted argument name; has_arg_ is for programmatic use and takes a string naming a argument.

See Also

hasArg.

Examples

has_arg(x, mean.default)
has_arg(y, mean.default)
f <- function(...) has_arg(z)
f(z = 123)
f(123)

assert_is_binding_locked

Is the binding of a variable locked?

Description

Check to see if the binding of a variable is locked (that is, it has been made read-only).

Usage

assert_is_binding_locked(x, severity = getOption("assertive.severity", "stop"))

is_binding_locked(x, env = if (is_scalar(e <- findNnameIII asNenvironment(eI else parentNframe(IL Nxname = get_name_in_parent(xII

Arguments

x Input to check. (Unlike bindingIsLocked, you can pass the variable itself, rather than a string naming that variable.)
severity How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
env Environment to check where binding had been locked.
.xname Not intended to be used directly.
assert_is_debugged

Value

TRUE or FALSE, depending upon whether or not the binding is locked in the specified environment. `assert_is_binding_locked` returns nothing but throws an error if the corresponding `is_*` function returns FALSE.

Note

The environment is guessed as follows: The name of `x` is determined via `get_name_in_parent`. Then `find` is called.

See Also

`bindingIsLocked`, which this wraps, `find` for how the environment is guessed. If this returns a single environment, that is used. Otherwise the parent environment is used (as determined with `parent.frame`).

Examples

```r
is_binding_locked(a_non_existent_variable)
res <- new.env()
res$x <- 1:10
is_binding_locked(x, res)
lockBinding("x", res)
is_binding_locked(x, res)
unlockBinding("x", res)
is_binding_locked(x, res)
```

assert_is_debugged

Is the input function being debugged?

Description

Checks to see if the input DLL (a.k.a. shared object) is loaded.

Usage

```r
assert_is_debugged(x, severity = getOption("assertive.severity", "stop"))
is_debugged(x, .xname = get_name_in_parent(x))
```

Arguments

- `x` Input to check.
- `severity` How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
- `.xname` Not intended to be used directly.
assert_is_if_condition

Value

is_debugged wraps isdebugged, providing more information on failure. assert_is_debugged returns nothing but throws an error if is_debugged returns FALSE.

See Also

isdebugged.

assert_is_if_condition

Is suitable to be used as an if condition

Description

Is suitable to be used as an if condition

Usage

assert_is_if_condition(x, severity = getOption("assertive.severity", "stop"))

is_if_condition(x, .xname = get_name_in_parent(x))

Arguments

x Input to check.
severity How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
.xname Not intended to be used directly.

Value

is_if_condition returns TRUE if the input is scalar TRUE or FALSE.

Note

if will try to do the right thing if you pass it a number or a string, but this function assumes you want to do the right thing and pass either TRUE or FALSE, maybe with some attributes.
assert_is_loaded

Is the input a symbol in a loaded DLL?

Description

Checks to see if the input DLL (a.k.a. shared object) is loaded.

Usage

assert_is_loaded(x, severity = getOption("assertive.severity", "stop"))

is_loaded(x, PACKAGE = "", type = c("", "C", "Fortran", "Call", "External"), .xname = get_name_in_parent(x))

Arguments

x A string naming the symbol in a DLL to check.

severity How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".

PACKAGE A string naming an R package to restrict the search to, or "" to check all packages. Passed to is.loaded.

type A string naming the type of external code call to restrict the search to, or "" to check all type. Passed to is.loaded.

.xname Not intended to be used directly.

Value

is_loaded wraps is_loaded, providing more information on failure. assert_is_loaded returns nothing but throws an error if is_loaded returns FALSE.

See Also

is.loaded.
assert_is_valid_r_code

\textit{Is the input valid R code?}

\section*{Description}
Check to see if the input is a valid (parseable) R code.

\section*{Usage}
\begin{verbatim}
assert_is_valid_r_code(x, severity = getOption("assertive.severity", "stop"))

is_valid_r_code(x, .xname = get_name_in_parent(x))
\end{verbatim}

\section*{Arguments}
\begin{itemize}
\item \textbf{x} \hspace{1cm} Input to check.
\item \textbf{severity} \hspace{1cm} How severe should the consequences of the assertion be? Either "stop", "warning", "message", or "none".
\item \textbf{.xname} \hspace{1cm} Not intended to be used directly.
\end{itemize}

\section*{Value}
TRUE if the input string is valid R code.

\section*{See Also}
\begin{verbatim}
pause
\end{verbatim}

\section*{Examples}
\begin{verbatim}
is_valid_r_code("x <- 1 + sqrt(pi)")
is_valid_r_code("x <- ")
is_valid_r_code("<- 1 + sqrt(pi)")
\end{verbatim}

is_error_free \hspace{1cm} \textit{Does the code run without throwing an error?}

\section*{Description}
Call the code inside a try block and report if an error was thrown.

\section*{Usage}
\begin{verbatim}
is_error_free(x)
\end{verbatim}
Arguments

x  Code to check.

Value

TRUE if the code runs without throwing an error. The result of running the code is contained in an attribute named "result".

Note

Note that this has the side effect of running the code contained in x.
Index

assert_all_are_existing, 2                  parent.frame, 6
assert_all_are_valid_variable_names, 3     parse, 9
assert_any_are_existing
  (assert_all_are_existing), 2
assert_any_are_valid_variable_names
  (assert_all_are_valid_variable_names), 3
assert_has_arg, 4
assert_is_binding_locked, 5
assert_is_debugged, 6
assert_is_if_condition, 7
assert_is_loaded, 8
assert_is_valid_r_code, 9
bindingIsLocked, 6
exists, 3
find, 6
has_arg (assert_has_arg), 4
has_arg_ (assert_has_arg), 4
hasArg, 5
is.loaded, 8
is_binding_locked
  (assert_is_binding_locked), 5
is_debugged (assert_is_debugged), 6
is_error_free, 9
is_existing (assert_all_are-existing), 2
is_if_condition
  (assert_is_if_condition), 7
is_loaded (assert_is_loaded), 8
is_valid_r_code
  (assert_is_valid_r_code), 9
is_valid_variable_name
  (assert_all_are_valid_variable_names), 3
isdebugged, 7
make.names, 4