Package 'testextra'

July 12, 2019

Type Package

Title Extract Test Blocks

Version 0.1.0

Maintainer Andrew Redd <andrew.redd@hsc.utah.edu>

Description A collection of testing enhancements and utilities.
Including utilities for extracting inline test blocks from
package source files.

License GPL-2

Encoding UTF-8

Language en-US

LazyData true

Imports assertthat, methods, parsetools, pkgcond, postlogic, purrr,
rlang, stringi, testthat, utils

Suggests covr, devtools, withr, rstudioapi, htmltools, shiny, yaml, DT

RoxygenNote 6.1.1

Collate 'extract_tests.R' 'catch_condition.R' 'inheritance.R'
'new_namespace.R' 'strings.R' 'util-testing.R' 'validity.R'
'coverage.R'

URL https://github.com/RDocTaskForce/testextra

BugReports https://github.com/RDocTaskForce/testextra/issues

NeedsCompilation no

Author Andrew Redd [aut, cre] (<https://orcid.org/000-0002-6149-2438>),
R Documentation Task Force [cph] (https://rdoctaskforce.github.io/),
R Consortium [fnd] (https://www.r-consortium.org)

Repository CRAN

Date/Publication 2019-01-18 22:30:03 UTC

R topics documented:

`addin_covr_file` *Add-in for* `covr_file`

Description

This allows for covr_file to be run from a menu in RStudio.

Usage

```
addin_covr_file()
```

`addin_extract_covr` *Add-in for Extract & Coverage*

Description

Add-in for Extract & Coverage

Usage

```
addin_extract_covr()
```

addin_test *RStudio add-ins*

Description

RStudio add-ins

Usage

```
addin_test()

addin_test_file()
```

catch_condition *Catch a condition for testing.*

Description

This function captures a condition object such as a warning or error, to allow for testing components and classes.

Usage

```
catch_condition(code)

catch_all_conditions(code)
```

Arguments

code	code to run that should assert a condition.

Examples

```
(cond <- catch_condition(stop("catch me.")))
class(cond)

my_fun <- function(){
    message("a message")
    warning("a warning")
    pkg_message("a package message", scope="test")
    pkg_warning("a package warning", scope="test")
    pkg_error("a package error", scope='test')
}
conditions <- catch_all_conditions(my_fun())
conditions$messages
conditions$warnings
conditions$error  # only one error can be caught at a time.
```

`class-expectations` *Class Expectations*

Description

These extend the testthat::expect_is to have finer grain tests.

Usage

```
expect_is_not(object, class, info = NULL, label = NULL)

expect_is_exactly(object, class, info = NULL, label = NULL)

expect_all_inherit(object, class, info = NULL, label = NULL)
```

Arguments

`object`	the object in question.
`class`	the expected class object is to be.
`info`	extra information to be included in the message (useful when writing tests in loops).
`label`	object label. When `NULL`, computed from deparsed object.

Functions

- `expect_is_not`: test that an object does **not** inherit from a class.
- `expect_is_exactly`: test that an object is exactly a specific class and not a child class.
- `expect_all_inherit`: test that all elements of a list inherit a given class.

See Also

Other class: `class-tests`

Examples

```
# Test to make sure an object is not of a class.
## Not run:
# will return an error.
expect_is_not(1L, "numeric")

## End(Not run)

# but this is fine.
expect_is_not('a', "numeric")

expect_is_exactly('a', "character")
```

class-tests *Enhanced Class Tests*

Description

These tests allow for mapped and enhanced tests regarding class.

Usage

```
all_inherit(lst, class, label = NULL)

are(lst, class)

is_exactly(object, class)

all_are_exactly(lst, class, label = NULL)
```

Arguments

lst	A list of objects to test
class	The class object is to be, or classes it is allowed to be.
label	object label. When NULL, computed from deparsed object.
object	An object to test

Functions

- all_inherit: Check if all elements of a list are or inherit from the given class. Uses base::inherits() to check inheritance.
- are: methods::is mapped over a vector. Similar to all_inherit but uses methods::is() for test. This manifests in S4 Virtual classes such as the 'ANY' class
- is_exactly: Test that an object is exactly a class; excludes inheritance.
- all_are_exactly: Version of is_exactly for all elements of a list.

See Also

Other class: class-expectations

Examples

```
lst <- list(1L, 2, TRUE)

# all_inherit uses `inherits`
all_inherit(lst, 'numeric')
all_inherit(lst, 'integer')
all_inherit(lst, 'ANY')
```

```
# are uses `is` so gets different results.
are(lst, "numeric")
are(lst, "integer")
are(lst, "ANY")

# is_exactly the class must match exactly
is_exactly(1L, "integer")
# no inheritance allowed
is_exactly(1L, "numeric")
```

`class0` *Extract class as a single string.*

Description

Extract class as a single string.

Usage

```
class0(x)
```

Arguments

x	any object.

`covr-rendering-single` *Rendering for single file report*

Description

These functions facilitate the creation of reports for coverage of a single file.

Usage

```
.renderSourceRow(line, source, coverage)

.renderSourceFile(lines, file = "source", highlight = TRUE)

.single_file_summary(file_stats)

.renderReport(coverage, report.file, dir = dirname(report.file),
  libdir = file.path(dir, "lib"))
```

Arguments

line, lines	Line(s) number
source	source file
coverage	The number of times covered
file	the file in question
highlight	Highlight the row.
file_stats	The coverage object for the file.
report.file	Where to output the HTML report.
dir	the base directory for the HTML output
libdir	Where to put html dependencies?

---

covr-single *Single File Coverage*

Description

These functions extract tests, run tests and create a report of the coverage for a single file.

Usage

```
file_coverage(file = rstudioapi::getSourceEditorContext()$path,
  pkg = ".", ...)

covr_file(coverage = file_coverage(), report.file = NULL,
  show.report = interactive())
```

Arguments

file	The file to extract test from and compute coverage.
pkg	The package file is associated with.
...	Arguments passed on to covr::file_coverage **source_files** Character vector of source files with function definitions to measure coverage **test_files** Character vector of test files with code to test the functions **line_exclusions** a named list of files with the lines to exclude from each file. **function_exclusions** a vector of regular expressions matching function names to exclude. Example print\\. to match print methods. **parent_env** The parent environment to use when sourcing the files.
coverage	Coverage returned from file_coverage().
report.file	Where to save the HTML report.
show.report	if the HTML report should be displayed.

Functions

- `file_coverage`: Extract tests and compute the coverage for the given file.
- `covr_file`: Create a report for a single

---

covr_files *Compute coverage for a group of files.*

Description

Compute coverage for a group of files.

Usage

```
covr_files(filter, pkg = ".", report = TRUE)
```

Arguments

`filter`	A regular expression filter to apply to the files from `pkg`.
`pkg`	The package to compute coverage for.
`report`	If a report should be constructed and shown.

---

expect_valid *Expect an S4 object is valid*

Description

Similar to `is_valid()` except designed to work in the `testthat::test_that()` framework.

Usage

```
expect_valid(object, complete = FALSE, info = NULL, label = NULL)
```

Arguments

`object`	an S4 object to test for validity
`complete`	logical; if `TRUE`, `validObject` is called recursively for each of the slots. The default is `FALSE`.
`info`	extra information to be included in the message (useful when writing tests in loops).
`label`	object label. When `NULL`, computed from deparsed object.

See Also

Other validity-tests: `validity-tests`

`extract_tests` *Extract tests from source*

Description

Use this function to extract tests from package source files. In-source testing blocks are contained in blocks that are prevented from running when sourced by an `if(FALSE){...}` statement. It also contains a documentation tag to denote a testing block.

Usage

```
extract_tests(pkg = ".", filter = NULL,
  verbose = getOption("verbose", FALSE), full.path = NA,
  force = FALSE)
```

Arguments

`pkg`	The root directory of the package.
`filter`	If specified, only tests from files matching this regular expression are extracted.
`verbose`	Print message?
`full.path`	Include full file paths in generated files. TRUE, indicates full path, FALSE, indicated only basename, and NA(default) implies path relative to `pkg`.
`force`	Force test extraction even if the generated test file is newer than the corresponding source file.

Details

The first line of the block should look similar to

```
if(FALSE){#@testing [optional information]
...
}
```

Examples

```
## Not run:
# Extract all files
extract_tests('.')

# Extract only files that start with 'Class-' or 'class-'
extract_tests('.', filter="^[Cc]lass-.*\\.[Rr]$")

## End(Not run)
```

is_valid_regex *Check if a regular expression is valid.*

Description

Check if a regular expression is valid.

Usage

```
is_valid_regex(pattern)
```

Arguments

pattern	the regular expression pattern to test.

namespaces *Create namespace environments*

Description

Create and manipulate namespace and test package environments.

Usage

```
new_namespace_env(name, path = file.path(tempdir()),
  import = "methods")

new_pkg_environment(name = "test package environment", ...,
  register = FALSE)

register_namespace(ns)

unregister_namespace(ns)

is_namespace_registered(ns)
```

Arguments

name	The name of the environment
path	An optional path.
import	Package to include in the imports.
...	Arguments passed on to new_namespace_env **name** The name of the environment **path** An optional path.

	import Package to include in the imports.
register	Should the package namespace be registered?
ns	a namespace environment or a character name of a namespace.

Functions

- new_namespace_env: Create a new namespace environment
- new_pkg_environment: Create a package environment. All package environments are namespaces but not all namespaces qualify as package environments.
- register_namespace: Register a namespace
- unregister_namespace: Remove a namespace from the registry
- is_namespace_registered: Check if a namespace is registered

Examples

```
ns <- new_namespace_env('my namespace')
isNamespace(ns)
environmentName(ns)
packageName(ns) # not a package

pkg <- new_pkg_environment("myPackage")
isNamespace(pkg)
environmentName(pkg)
packageName(pkg)             # now a package
is_namespace_registered(pkg) # but not registered
## Not run:
asNamespace("myPackage")     # so this WILL NOT work.

## End(Not run)

register_namespace(pkg)
is_namespace_registered(pkg) # now registered
asNamespace("myPackage")     # so this WILL work.

unregister_namespace(pkg)
is_namespace_registered(pkg) # now unregistered
isNamespace(pkg)             # but still a namespace
```

---

string-tests *Tests for strings*

Description

Tests for strings

Usage

```
is_nonempty_string(x)

is_optional_string(x)
```

Arguments

x	a character vector/string.

Functions

- `is_nonempty_string`: Test that a character is both a string (character vector of length one) and that it is non-empty, has at least one character and is not missing.
- `is_optional_string`: Check for an optional string: must be a character, not missing, a vector of either length 0 or 1, and if provided must not be empty ("").

Examples

```
# TRUE
is_nonempty_string("hello")

# All FALSE
x <- c("hello", "world")
is_nonempty_string(x)
is_nonempty_string(NA_character_)
is_nonempty_string(character(0))
is_nonempty_string(NULL)
is_nonempty_string(12345)
```

---

test *Extract and run package tests*

Description

This function corresponds to an intentionally masks `devtools::test()` from the `devtools` package. This version is polymorphic depending on the number of arguments given.

Usage

```
test(..., pkg = switch(nargs(), ".", ..1), filter = switch(...length(),
  ..1, ..2))

extract_and_test_file(file = rstudioapi::getSourceEditorContext()$path,
  pkg = rstudioapi::getActiveProject())
```

Arguments

`...`	polymorphic arguments
`pkg`	The package to test.
`filter`	An optional filter to restrict the files to extract from and run tests for.
`file`	for `test_file` the exact file to extract and test from.

Details

When no arguments are provided all tests are extracted and run from the package corresponding to the active working directory. In other words `test()` is equivalent to `test(pkg='.', filter=NULL)`

If arguments are provided they may be named. If any argument is named all must be named, if not found the two key parameters will be taken to be

Examples

```
## Not run:
# Extract and run all tests for the package in the
# current working directory.
test()

# One argument form
# extract and test class files for the
# package in the current working directory.
test("^Class-")

# Two argument form
# Extract files matching "Class" in the filename
# for the package located at "inst/textExtractionTest"
test("inst/testExtractionTest", "Class")


## End(Not run)
```

`validity-tests` *Alternate check for validity*

Description

These functions will test if an object is valid returning a value appropriate to use in `assertthat::validate_that()`, `assertthat::assert_that()`, or `assertthat::see_if()`.

Usage

```
is_valid(object, complete = FALSE)

are_valid(lst, complete = FALSE)
```

Arguments

object	an S4 object to test for validity
complete	logical; if TRUE, validObject is called recursively for each of the slots. The default is FALSE.
lst	a list of S4 objects to test for validity.

Functions

- is_valid: Check if an object is valid.
- are_valid: Check if each element in a list is valid.

See Also

Other validity-tests: expect_valid

Index