Package: RUnit (via r-universe)

November 14, 2026

Version 0.4.33

Date 2024-02-09

Title R Unit Test Framework

Author Matthias Burger <burgerm@users.sourceforge.net>, Klaus Juenemann <k.junemann@gmx.net>, Thomas Koenig <thomas.koenig@epigenomics.com>

Maintainer Roman Zenka <zenka.roman@mayo.edu>

LazyLoad yes

Depends R (>= 2.5.0), utils (>= 2.5.0), methods (>= 2.5.0), graphics (>= 2.5.0)

Suggests XML (>= 3.1.0)

Description R functions implementing a standard Unit Testing framework, with additional code inspection and report generation tools.

License GPL-2

NeedsCompilation no

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

Date/Publication 2024-02-22 17:05:08 UTC

RemoteUrl https://github.com/romanzenka/runit

RemoteRef HEAD

RemoteSha 5eb24a95c4c0a7690996783e275d316c43a2f79d

Contents

.setUp 2
checkFuncs 3
inspect 5
options 7
printHTML 8
RUnit 9

runTestSuite 10
textProtocol 13
tracker 16

Index **18**

`.setUp` *Definition of RUnit Test Case code files.*

Description

Either one or both functions have to be provided by the test case author, take precedence over the dummy definitions provided by the RUnit package and are called once for every test case identified.

Usage

```
.setUp()
.tearDown()
```

Details

To be written ...

Value

Functions do not return a value; called for their side effects.

Author(s)

Thomas König, Klaus Jünemann & Matthias Burger

See Also

`runTestFile`.

checkFuncs *RUnit check functions*

Description

A set of functions used to check the results of some test calculation. If these functions are called within the RUnit framework, the results of the checks are stored and reported in the test protocol.

`checkEquals` compares two R objects by invoking `all.equal` on the two objects. If the objects are not equal an error is generated and the failure is reported to the test logger such that it appears in the test protocol.

`checkEqualsNumeric` works just like `checkEquals` except that it invokes `all.equal.numeric` instead of `all.equal`

`checkIdentical` is a convenience wrapper around identical using the error logging mechanism of RUnit.

`checkTrue` uses the function `identical` to check if the expression provided as first argument evaluates to `TRUE`. If not, an error is generated and the failure is reported to the test logger such that it appears in the test protocol.

`checkException` evaluates the passed expression and uses the `try` mechanism to check if the evaluation generates an error. If it does the test is OK. Otherwise an error is generated and the failure is reported to the test logger such that it appears in the test protocol.

`DEACTIVATED` interrupts the test function and reports the test case as deactivated. In the test protocol deactivated test functions are listed separately. Test case deactivation can be useful in the case of major refactoring. Alternatively, test cases can be commented out completely but then it is easy to forget the test case altogether.

Usage

```
checkEquals(target, current, msg,
            tolerance = .Machine$double.eps^0.5,
            checkNames = TRUE, ...)
checkEqualsNumeric(target, current, msg,
                   tolerance = .Machine$double.eps^0.5, ...)
checkIdentical(target, current, msg)
checkTrue(expr, msg)
checkException(expr, msg, silent = getOption("RUnit")$silent)
DEACTIVATED(msg)
```

Arguments

`current`, `target`	objects to be compared (`checkEqualsNumeric` cannot handle S4 class objects).
`msg`	an optional message to document a check and to facilitate the identification of a possible failure. The message only appears as text in the test protocol, it is not further used in any of the check functions.
`tolerance`	numeric >= 0. A numeric check does not fail if differences are smaller than 'tolerance'.

checkNames	flag, if FALSE the names attributes are set to NULL for both current and target before performing the check.
expr	syntactically valid R expression which can be evaluated and must return a logical scalar (TRUE|FALSE). A named expression is also allowed but the name is disregarded.
silent	flag passed on to try, which determines if the error message generated by the checked function is displayed. Queried from global options set for RUnit at package load.
...	optional arguments passed to all.equal or all.equal.numeric

Details

The check functions are direct equivalents of the various methods of the class junit.framework.Assert of Javas JUnit framework which served as basis for the RUnit package.

For functions defined inside a package equipped with a namespace only exported functions can be accessed inside test cases directly. For functions not exported the only way to test them is to use the ':::' operator combined with the package name as a prefix.

Special care is required if test cases are written for S4 classes and methods. If a new class is defined inside a test case via a setClass call the class is added to the global class cache and thus available outside the test case. It will persist until explicitly removed via a removeClass call. Same applies for new method and generic definitions. Be sure to remove methods and classes in each test case they are defined after the checks have been performed. This is an advise gained from the cumbersome experience: not doing so leads to difficult to pin down error causes incurred from previously executed test cases. For a simple example see the provided test cases in /github/workspace/pkglib/RUnit/examples/runitVirtualClassTest.r.

Author(s)

Thomas König, Klaus Jünemann & Matthias Burger

See Also

all.equal, all.equal.numeric and identical are the underlying comparison functions. try is used for error catching. .setUp for details on test case setup. See RUnit-options for global options controlling log out.

Examples

```
checkTrue(1 < 2, "check1")     ## passes fine
## checkTrue(1 > 2, "check2")  ## appears as failure in the test protocol

v <- 1:3
w <- 1:3
checkEquals(v, w)               ## passes fine
names(v) <- c("A", "B", "C")
## checkEquals(v, w)            ## fails because v and w have different names
checkEqualsNumeric(v, w)        ## passes fine because names are ignored
```

```
x <- rep(1:12, 2)
y <- rep(0:1, 12)
res <- list(a=1:3, b=letters, LM=lm(y ~ x))
res2 <- list(a=seq(1,3,by=1), b=letters, LM=lm(y ~ x))
checkEquals( res, res2)        ## passes fine
checkIdentical( res, res)
checkIdentical( res2, res2)
## checkIdentical( res, res2)  ## fails because element 'a' differs in type


fun <- function(x) {
  if(x)
  {
   stop("stop conditions signaled")
  }
  return()
}

checkException(fun(TRUE))      ## passes fine
## checkException(fun(FALSE))  ## failure, because fun raises no error
checkException(fun(TRUE), silent=TRUE)

##  special constants
##  same behaviour as for underlying base functions
checkEquals(NA, NA)
checkEquals(NaN, NaN)
checkEquals(Inf, Inf)

checkIdentical(NA, NA)
checkIdentical(NaN, NaN)
checkIdentical(-Inf, -Inf)

## DEACTIVATED("here one can document on the reason for deactivation")
```

`inspect` *Track the executed code lines of a function or method.*

Description

`inspect` examines and modifies the source code of a function or method. After the modification of the source code, the modified function will be executed and the result of the tracking process will be stored. To store the information a `tracker` environment with the name track must exist. Note, that not all R code constructs can be handled at the current state. In some cases it is not possible to track a specific code line. Therefore, clearly structured code with consequent use of opening and closing braces to indicate conditional expressions can prevent these parser problems.

Usage

```
inspect(expr, track = track)
```

Arguments

expr	Any R function or method call.
track	list object, as returned by a call to tracker.

Details

The return value of inspect is the result returned by the function executed. If the function has no return value nothing is returned either.

Author(s)

Thomas König, Klaus Jünemann & Matthias Burger

See Also

tracker for the call tracking object, and printHTML for displaying results.

Examples

```
## example function
foo <- function(x){
   y <- 0
   for(i in 1:100)
   {
      y <- y + i
   }
   return(y)
}

## the name track is necessary
track <- tracker()

## initialize the tracker
track$init()

## inspect the function
## res will collect the result of calling foo
res <- inspect(foo(10), track = track)

## get the tracked function call info
resTrack <- track$getTrackInfo()

## create HTML sites
printHTML(resTrack, baseDir=tempdir())
```

options *RUnit options*

Description

RUnit uses three options available via the global R options list

Details

RUnit specif options are added to R's global options list on package loading and removed again on pachage unloading.

Options used in RUnit

`silent`: logical flag, default `FALSE`, sets the 'silent' argument for `checkException`. Allows to globally silence output from exception checks for all test suites excuted in one run.

`verbose`: non-negative integer, default `1`, `0`: surpresses enclosing begin/end messages for each test case, `1`: output enclosing begin/end messages for each test case

`outfile`: `NULL`, connection or character, default `NULL`. If non-null has to be an open connection or a file name. Will be used to redirect all output to specified file/connection using `sink`. Connection is close after test suite execution call (via `runTestSuite` or `runTestFile`) has completed. If the file exists it is overwriten.

Author(s)

Matthias Burger

See Also

`options`, `getOption`, `sink`.

Examples

```
## Not run:
##  quiet log output
ro <- getOption("RUnit")
ro$silent <- TRUE
ro$verbose <- 0L
options("RUnit"=ro)

## End(Not run)
```

printHTML *Write HTML pages of the tracking result.*

Description

printHTML creates a subdirectory named "result" in the base directory specified via baseDir. All HTML pages and images will be put in that directory.

Usage

```
printHTML(object, baseDir = ".")
```

Arguments

object	'trackInfo' S3 class object (list), containing the result of the function tracker.
baseDir	A character string, specifying the base directory for the HTML pages to be written to. Defaults to the current working directory.

Details

An "index.html" page will be created in the directory "results" which is the root entry page of the HTML pages. The displayed result for every tracked function consists of two HTML pages. The first page is an overview on how often every line of code was executed. Code lines not executed are highlighted red, executed lines are shown in green. The second page is a graph representation of the execution flow of the function. Each code line has a edge pointing to the next code line that is executed subsequently. Thus loops and jumps become clearly visible.

Author(s)

Thomas König, Klaus Jünemann & Matthias Burger

See Also

tracker for the call tracking object definition.

Examples

```
##  example function
foo <- function(x){
   y <- 0
   for(i in 1:100)
   {
      y <- y + i
   }
   return(y)
}

##  the name track is necessary
```

```
track <- tracker()

##  initialize the tracker
track$init()

##  inspect the function
##  res is the result of foo
res <- inspect(foo(10), track = track)

##  get the tracking info
resTrack <- track$getTrackInfo()

##  create HTML pages
printHTML(resTrack, baseDir=tempdir())
```

RUnit *RUnit - Package Description*

Description

This package models the common Unit Test framework for R and provides functionality to track results of test case execution and generate a summary report. It also provides tools for code inspection and thus for test case coverage analysis. The design is inspired by the popular JUnit unit test framework.

This package comes with a set of unit tests, serving as a test battery to check correct functioning against new R versions released as well as practical examples for writing test cases (see the 'inst/unitTests' subdirectory of the source package, or 'unitTests' contained in the binary package version).

Author(s)

Thomas König, Klaus Jünemann & Matthias Burger

References

RUnit - A Unit Test Framework for R. useR! 2004 Vienna

See Also

See defineTestSuite, runTestSuite for unit testing or inspect and tracker for code inspection.

runTestSuite *Definition and execution of RUnit test suites.*

Description

runTestSuite is the central function of the RUnit package. Given one or more test suites it identifies and sources specified test code files one after another and executes all specified test functions defined therein. This is done sequentially for suites, test code files and test functions. During the execution information about the test function calls including the possible occurrence of failures or errors is recorded and returned at the end of the test run. The return object can then be used to create a test protocol of various formats.

runTestFile is just a convenience function for executing the tests in a single test file.

defineTestSuite is a helper function to define a test suite. See below for a precise definition of a test suite.

isValidTestSuite checks if an object defines a valid test suite.

Usage

```
defineTestSuite(name, dirs, testFileRegexp = "^runit.+\\.[rR]$",
                testFuncRegexp = "^test.+",
                rngKind = "Marsaglia-Multicarry",
                rngNormalKind = "Kinderman-Ramage")
isValidTestSuite(testSuite)
runTestSuite(testSuites, useOwnErrorHandler = TRUE,
             verbose = getOption("RUnit")$verbose,
             gcBeforeTest = FALSE)
runTestFile(absFileName, useOwnErrorHandler = TRUE,
            testFuncRegexp = "^test.+",
            rngKind = "Marsaglia-Multicarry",
            rngNormalKind = "Kinderman-Ramage",
            verbose = getOption("RUnit")$verbose,
            gcBeforeTest = FALSE)
```

Arguments

name	The name of the test suite.
dirs	Vector of absolute directory names where to look for test files.
testFileRegexp	Regular expression for matching test files.
testFuncRegexp	Regular expression for matching test functions.
rngKind	name of an available RNG (see RNGkind for possible options).
rngNormalKind	name of a valid rnorm RNG version (see RNGkind for possible options).
testSuite	A single object of class test suite.
testSuites	A single object of class test suite or a list of test suite objects.

`useOwnErrorHandler`
: If `TRUE` the RUnit framework installs its own error handler during test case execution (but reinstalls the original handler before it returns). If `FALSE` the error handler is not touched by RUnit but then the test protocol does not contain any call stacks in the case of errors.

`verbose`
: level of verbosity of output log messages, 0: omits begin/end comments for each test function. Queried from global options set for RUnit at package load.

`absFileName`
: Absolute file name of a test function.

`gcBeforeTest`
: Run garbage collector before executing a test for more precise test timing. Enabling this option makes the tests run longer, especially when testing many small tests. By default GC is disabled (since 0.4.32).

Details

The basic idea of the RUnit test framework is to declare a certain set of functions to be test functions and report the results of their execution. The test functions must not take any parameter nor return anything such that their execution can be automatised.

The specification which functions are taken as test functions is contained in an object of class `RUnitTestSuite` which is a list with the following elements.

name A simple character string. The name of a test suite is mainly used to create a well structure test protocol.

dirs A character vector containing the absolute names of all directories where to look for test files.

testFileRegexp A regular expression specifying the test files. All files in the test directories whose names match this regular expression are taken as test files. Order of file names will be alphabetical but depending on the used locale.

testFuncRegexp A regular expression specifying the test functions. All functions defined in the test files whose names match this regular expression are used as test functions. Order of test functions will be alphabetical.

After the RUnit framework has sequentially executed all test suites it returns all data collected during the test run as an object of class `RUnitTestData`. This is a (deeply nested) list with one list element for each executed test suite. Each of these executed test suite lists contains the following elements:

nTestFunc The number of test functions executed in the test suite.

nErr The number of errors that occurred during the execution.

nFail The number of failures that occurred during the execution.

dirs The test directories of the test suite.

testFileRegexp The regular expression for identifying the test files of the test suite.

testFuncRegexp The regular expression for identifying the test functions of the test suite.

sourceFileResults A list containing the results for each separate test file of the test suite.

The `sourceFileResults` list just mentioned contains one element for each specified test function in the source file. This element is a list with the following entries:

kind Character string with one of `success`, `error` or `failure` describing the outcome of the test function.

msg the error message in case of an error or failure and `NULL` for a successfully executed test function.

time The duration (measured in seconds) of the successful execution of a test function and `NULL` in the case of an error or failure. When running with `gcBeforeTest` option set to `FALSE` (default since 0.4.32), the timing of the tests might be misleading when garbage collector has to reclaim memory allocated by a previous test.

traceBack The full trace back as a character vector in the case of an error and `NULL` otherwise.

To further control test case execution it is possible to define two parameterless function `.setUp` and `.tearDown` in each test file. `.setUp()` is executed directly before and `.tearDown()` directly after each test function execution.

Quite often, it is useful to base test cases on random numbers. To make this procedure reproducible, the function `runTestSuite` sets the random number generator to the default setting `RNGkind(kind="Marsaglia-Multicarry normal.kind="Kinderman-Ramage")` before sourcing each test file (note that this default has been chosen due to historical reasons and differs from the current R default). This default can be overwritten by configuring the random number generator at the beginning of a test file. This setting, however, is valid only inside its own source file and gets overwritten when the next test file is sourced.

Value

`runTestSuite` and `runTestFile` both return an object of class RUnitTestData.

`defineTestSuite` returns an object of class `RUnitTestSuite`.

Author(s)

Thomas König, Klaus Jünemann & Matthias Burger

See Also

`checkTrue` and friends for writing test cases. `printTextProtocol` and `printHTMLProtocol` for printing the test protocol. See RUnit-options for global options controlling log out.

Examples

```
## run some test suite
myTestSuite <- defineTestSuite("RUnit Example",
                               system.file("examples", package = "RUnit"),
                               testFileRegexp = "correctTestCase.r")
testResult <- runTestSuite(myTestSuite)

##  same but without the logger being involved
## source(file.path(system.file("examples", package = "RUnit"),
##                  "correctTestCase.r"))
## test.correctTestCase()
```

```
## prints detailed text protocol
## to standard out:
printTextProtocol(testResult, showDetails = TRUE)

##  use current default RNGs
myTestSuite1 <- defineTestSuite("RUnit Example",
                                system.file("examples", package = "RUnit"),
                                testFileRegexp = "correctTestCase.r",
                                rngKind = "Mersenne-Twister",
                                rngNormalKind = "Inversion")

testResult1 <- runTestSuite(myTestSuite)


##  for single test files, e.g. outside a package context
testResult2 <- runTestFile(file.path(system.file("examples",
                                                 package = "RUnit"),
                                     "correctTestCase.r"))
printTextProtocol(testResult2, showDetails = TRUE)
```

`textProtocol` *Printing a plain text, HTML or JUnit-like XML version of an RUnit test run protocol.*

Description

`printTextProtocol` prints a plain text protocol of a test run. The resulting test protocol can be configured through the function arguments.

`printHTMLProtocol` prints an HTML protocol of a test run. For long outputs this version of the test protocol is slightly more readable than the plain text version due to links in the document. The resulting test protocol can be configured through the function arguments.

`printJUnitProtocol` prints a JUnit-style XML protocol of a test run. This feature is especially useful when running your RUnit tests through a continuous integration server that understands JUnit (like Jenkins). This machine-parseable output allows you to track errors over time, sort by execution time and similar useful tricks. To take advantage of this function, you have to have the XML package installed.

`print` prints the number of executed test functions and the number of failures and errors.

`summary` directly delegates the work to `printTextProtocol`.

`getErrors` returns a list containing the number of test functions, the number of deactivated functions (if there are any), the number of errors and the number of failures.

Usage

```
printTextProtocol(testData, fileName = "",
                  separateFailureList = TRUE,
                  showDetails = TRUE, traceBackCutOff = 9)
printHTMLProtocol(testData, fileName = "",
```

```
                    separateFailureList = TRUE,
                    traceBackCutOff = 9,
                    testFileToLinkMap = function(x) x )
  printJUnitProtocol(testData, fileName = "")
  ## S3 method for class 'RUnitTestData'
print(x, ...)
  ## S3 method for class 'RUnitTestData'
summary(object, ...)
  getErrors(testData)
```

Arguments

`testData`, `x`, `object`
: objects of class `RUnitTestData`, typically obtained as return value of a test run.

`fileName`
: Connection where to print the text protocol (printing is done by the `cat` command).

`separateFailureList`
: If `TRUE` a separate list of failures and errors is produced at the top of the protocol. Otherwise, the failures and errors are only listed in the details section.

`showDetails`
: If `TRUE` the protocol contains a detailed listing of all executed test functions.

`traceBackCutOff`
: The details section of the test protocol contains the call stack for all errors. The first few entries of the complete stack typically contain the internal RUnit function calls that execute the test cases and are irrelevant for debugging. This argument specifies how many calls are removed from the stack before it is written to the protocol. The default value is chosen such that all uninteresting RUnit calls are removed from the stack if `runTestSuite` has been called from the console. This argument takes effect only if `showDetails=TRUE`.

`testFileToLinkMap`
: This function can be used to map the full name of the test file to a corresponding html link to be used in the html protocol. By default, this is the identity map. See example below.

`...`
: additional arguments to summary are passed on to the printTextProtocol() call.

Details

The text protocol can roughly be divided into three sections with an increasing amount of information. The first section as an overview just reports the number of executed test functions and the number of failures and errors. The second section describes all test suites. Optionally, all errors and failures that occurred in some test suite are listed. In the optional third section details are given about all executed test functions in the order they were processed. For each test file all test functions executed are listed in the order they were executed. After the test function name the number of `check<*>` function calls inside the test case and the execution time in seconds are stated. In the case of an error or failure as much debug information as possible is provided.

Author(s)

Thomas König, Klaus Jünemann, Matthias Burger & Roman Zenka

See Also

runTestSuite

Examples

```
## run some test suite
myTestSuite <- defineTestSuite("RUnit Example",
                               system.file("examples", package = "RUnit"),
                               testFileRegexp = "correctTestCase.r")
testResult <- runTestSuite(myTestSuite)


## prints detailed text protocol
## to standard out:
printTextProtocol(testResult, showDetails = TRUE)
## prints detailed html protocol
## to standard out
printHTMLProtocol(testResult)
## prints JUnit-style XML protocol
## to standard out.
## You need to have XML package installed for this
if(require("XML")) {
  printJUnitProtocol(testResult)
}

## Not run:
##  example function to add links to URL of the code files in a code
##  repository, here the SourceForge repository
testFileToSFLinkMap <- function(testFileName, testDir = "tests") {
    ##  get unit test file name
    bname <- basename(testFileName)

    ## figure out package name
    regExp <- paste("^.*/([\.a-zA-Z0-9]*)/", testDir,"/.*$", sep = "")
    pack <- sub(regExp, "\1", testFileName)
    return(paste("http://runit.cvs.sourceforge.net/runit/",
                 pack, testDir, bname, sep = "/"))
  }


##  example call for a test suite run on the RUnit package
testSuite <- defineTestSuite("RUnit", "<path-to-source-folder>/RUnit/tests",
                             testFileRegexp = "^test.+")
testResult <- runTestSuite(testSuite)
printHTMLProtocol(testResult, fileName = "RUnit-unit-test-log.html",
                  testFileToLinkMap = testFileToSFLinkMap )

## End(Not run)
```

`tracker` *Tracking the results of the inspect process.*

Description

The current implementation uses the 'closure trick' to hide all details from the user and only allows to retrieve the results of the code inspection. `tracker` is used to create a new environment to manage and store the results of the tracking process. The `inspect` function requires such an environment with the name "track" (currently mandatory). The tracker records how often each and every function was called by `inspect` and summarizes the results of all calls. `tracker$init` initializes the tracker environment. `tracker$getTrackInfo` returns a list with the tracked results of the inspection process.

Usage

```
tracker()
```

Details

The 'trackInfo' S3 class object (list) has one entry for each function on the inspect list with the following elements:

src The source code of the function.

run The number of executions for each line of code.

graph A matrix. Each element in the matrix counts how often a code line was called from the previous code line in the execution flow.

nrRuns Counts how often the function was called.

funcCall The declaration of the function.

Methods

`init`	initializes the tracker environment
`addFunc`	add function to the inspect tracking list (internal use)
`getSource`	return the modified source code used for during inspection the specified index (internal use)
`bp`	update tracking info for specified function index (internal use)
`getTrackInfo`	return 'trackInfo' object
`isValid`	check 'trackInfo' object for conformance to class contract

Author(s)

Thomas König, Klaus Jünemann & Matthias Burger

See Also

inspect for the registration of functions & methods to be on the tracking list, and printHTML for displaying results

Examples

```
## example functions
foo <- function(x){
   y <- 0
   for(i in 1:100)
   {
      y <- y + i
   }
   return(y)
}

bar <- function(x){
   y <- 0
   for(i in 1:100)
   {
      y <- y - i
   }
   return(y)
}



##  the object name track is 'fixed' (current implementation)
track <- tracker()

##  initialize the tracker
track$init()

##  inspect the function
##  resFoo1 will contain the result of calling foo(50)
resFoo1 <- inspect(foo(50), track = track)

resFoo2 <- inspect(foo(20), track = track)

resBar1 <- inspect(bar(30), track = track)

##  get the tracked function call info for all inspect calls
resTrack <- track$getTrackInfo()

##  create HTML sites in folder <tempdir>/results for all inspect calls
printHTML(resTrack, baseDir=tempdir())
```

Index

∗ **RUnit**
  .setUp, 2
  checkFuncs, 3
  inspect, 5
  options, 7
  printHTML, 8
  RUnit, 9
  runTestSuite, 10
  textProtocol, 13
  tracker, 16
∗ **environment**
  options, 7
∗ **programming**
  .setUp, 2
  checkFuncs, 3
  inspect, 5
  options, 7
  printHTML, 8
  RUnit, 9
  runTestSuite, 10
  textProtocol, 13
  tracker, 16
∗ **test runner**
  runTestSuite, 10
∗ **test suite**
  runTestSuite, 10
.setUp, 2, *4*
.tearDown, *12*
.tearDown *(*.setUp*)*, 2
:::, *4*

all.equal, *4*
all.equal.numeric, *4*

checkEquals *(*checkFuncs*)*, 3
checkEqualsNumeric *(*checkFuncs*)*, 3
checkException *(*checkFuncs*)*, 3
checkFuncs, 3
checkIdentical *(*checkFuncs*)*, 3
checkTrue, *12*
checkTrue *(*checkFuncs*)*, 3

DEACTIVATED *(*checkFuncs*)*, 3
defineTestSuite, *9*
defineTestSuite *(*runTestSuite*)*, 10

getErrors *(*textProtocol*)*, 13
getOption, *7*

identical, *4*
inspect, 5, *9*, *17*
isValidTestSuite *(*runTestSuite*)*, 10

options, 7, *7*

print.RUnitTestData *(*textProtocol*)*, 13
printHTML, *6*, 8, *17*
printHTMLProtocol, *12*
printHTMLProtocol *(*textProtocol*)*, 13
printJUnitProtocol *(*textProtocol*)*, 13
printTextProtocol, *12*
printTextProtocol *(*textProtocol*)*, 13

removeClass, *4*
RNGkind, *10*
RUnit, 9
RUnit options *(*options*)*, 7
RUnit-options, *4*, *12*
RUnit-options *(*options*)*, 7
runTestFile, *2*
runTestFile *(*runTestSuite*)*, 10
runTestSuite, *9*, 10, *15*

setClass, *4*
sink, *7*
summary.RUnitTestData *(*textProtocol*)*, 13

textProtocol, 13
tracker, *6*, *8*, *9*, 16
try, *4*