

# Package ‘tinytest2JUnit’

May 30, 2024

**Type** Package

**Title** Convert 'tinytest' Output to JUnit XML

**Version** 1.1.2

**Maintainer** Lennart Tuijnder <lennart.tuijnder@openanalytics.eu>

**Description** Unit testing is a solid component of automated CI/CD pipelines. 'tinytest' - a lightweight, zero-dependency alternative to 'testthat' was developed. To be able to integrate 'tinytests' results into common CI/CD systems the test results from tinytest need to be captured and converted to JUnit XML format. 'tinytest2JUnit' enables this conversion while staying also lightweight and only have 'tinytest' as its dependency.

**Imports** tools, parallel, utils, tinytest

**URL** <https://github.com/openanalytics/tinytest2JUnit>

**BugReports** <https://github.com/openanalytics/tinytest2JUnit/issues>

**License** GPL-3

**Copyright** Open Analytics NV, 2024

**RoxygenNote** 7.3.1

**Encoding** UTF-8

**NeedsCompilation** no

**Author** Anne-Katrin Hess [aut],  
Lennart Tuijnder [aut, cre]

**Repository** CRAN

**Date/Publication** 2024-05-30 07:00:03 UTC

## R topics documented:

charVecToSingleLength . . . . .	2
classnameTestcase . . . . .	3
constructFailureDescription . . . . .	3
constructTestcaseTag . . . . .	4
constructTestsuitesTag . . . . .	4

constructTestsuiteTag . . . . .	5
errorTestcaseTag . . . . .	5
escapeXml . . . . .	6
escapeXmlText . . . . .	6
failureTestcaseTag . . . . .	7
format.XMLtag . . . . .	7
formattedFrame . . . . .	8
getFormattedStacktrace . . . . .	9
isSingleLengthCharNonNA . . . . .	9
nameTestcase . . . . .	10
passedTestcaseTag . . . . .	10
print.XMLtag . . . . .	11
runTestDir . . . . .	11
runTestFile . . . . .	13
sideeffectTestcaseTag . . . . .	14
tag . . . . .	14
testPackage . . . . .	15
writeJUnit . . . . .	17
[.tinytests2JUnit . . . . .	18

<b>Index</b>	<b>20</b>
--------------	-----------

---

charVecToSingleLength *Convert any will character vector to a single length character vector*

---

## Description

Convert any will character vector to a single length character vector

## Usage

```
charVecToSingleLength(x)
```

## Arguments

x                    a character

## Value

x a single-length character vector Non-NA

## Examples

```
tinytest2JUnit::charVecToSingleLength(c("Hello", "World")) # -> "HelloWorld"
tinytest2JUnit::charVecToSingleLength(c("Hello", NA_character_)) # -> "HelloNA"
tinytest2JUnit::charVecToSingleLength(character(0L)) # -> ""
```

---

classnameTestcase	<i>Helper function specifying the 'classname' attribute of the testcase tag.</i>
-------------------	--

---

**Description**

Helper function specifying the 'classname' attribute of the testcase tag. Currently equal to the fileName. The classname is already xml escaped.

**Usage**

```
classnameTestcase(tinytest)
```

**Arguments**

tinytest          a tinytest-object representing an individual test case.

**Value**

character(1) being the 'classname'

---

constructFailureDescription	<i>Helper function generating the body of a failure description tag!</i>
-----------------------------	--

---

**Description**

Helper function generating the body of a failure description tag! Attempts to mimic the print behaviour of a tinytest object.

**Usage**

```
constructFailureDescription(tinytest)
```

**Arguments**

tinytest          A tinytest objec that is considered failed!.

**Value**

character(1) being the failure tag description body. This string is already properly xml escaped.

---

```
constructTestcaseTag Construct JUnit </testcase> tag
```

---

**Description**

Construct JUnit </testcase> tag based on a single tinytest result.

**Usage**

```
constructTestcaseTag(tinytest)
```

**Arguments**

tinytest            a tinytest-object representing an individual test case.

**Value**

XML tag: with tag-name = tinytest and contains the test result per test.

---

```
constructTestsuitesTag  
Construct the JUnit </testsuites> tag
```

---

**Description**

Convert the tinytests2JUnit or tinytests-object containing test across possibly multiple files into a JUnit </testsuites> tag. More details are reported to the JUnit if a tinytests2JUnit object compared to the native tinytests object.

**Usage**

```
constructTestsuitesTag(testResults)
```

**Arguments**

testResults        tinytests2JUnit | tinytestsobject to convert into a JUnit XML object Usually the result of produced by `tinytest::test_package()` or `tinytest::run_test_dir()`.

**Details**

Reference for JUnit XML format: <https://llg.cubic.org/docs/junit/>

See details `runTestDir()` which additional info is recorded.

**Value**

XML tag: with tag-name = </testsuites>. This is the root of the JUnit XML document.

---

constructTestsuiteTag *Construct JUnit </testsuite> tag*

---

### Description

Construct the </testsuite> tag of a tinytest, given all the tinytest results from a single test file.

### Usage

```
constructTestsuiteTag(testsFile, id)
```

### Arguments

testsFile	tinytests2JUnit   tinytests -object with all test results of a specified test file. At least a single test is expected.
id	integer(1) testsuite id.

### Details

In case a tinytest2JUnit is provided following additional info can be reported:

- testsuite duration.
- timestamp when the testsuite was performed.
- hostname where the testsuite was ran.

### Value

XML tag: with tag-name = </testsuite> that contains all the test results per test file.

---

errorTestcaseTag *Construct a testcase-tag for an error test.*

---

### Description

Construct a testcase-tag for an error test.

### Usage

```
errorTestcaseTag(tinytest)
```

### Arguments

tinytest	a tinytest object already validated to be a "ERROR" test.
----------	---

### Value

a testase XML tag

escapeXml

*Escape xml*

---

**Description**

Escape the characters &,"',<,>

**Usage**

```
escapeXml(x)
```

**Arguments**

x                    a character vector meant to be xml

**Value**

The same character vector x but xml escaped.

**See Also**

<https://stackoverflow.com/a/1091953/10415129>

---

escapeXmlText

*Escape xml text*

---

**Description**

Escape the characters '<' and & in a character vector meant to be xml-text content.

**Usage**

```
escapeXmlText(x)
```

**Arguments**

x                    a character vector meant to be xml-text content.

**Value**

The same character vector x but xml text escaped.

---

failureTestcaseTag	<i>Construct a testcase-tag for a failed test.</i>
--------------------	--

---

**Description**

Construct a testcase-tag for a failed test.

**Usage**

```
failureTestcaseTag(tinytest)
```

**Arguments**

tinytest	a tinytest object already validated to be a "FAILURE" test.
----------	---

**Value**

a testase XMLtag

---

format.XMLtag	<i>Format method for XMLtag class</i>
---------------	---------------------------------------

---

**Description**

Format S3 method for the XMLtag-class

**Usage**

```
## S3 method for class 'XMLtag'
format(x, level = 0, ...)
```

**Arguments**

x	an XMLtag-object
level	print depth level. For each level 2 spaces are added to the left. The content of a tag is automatically indented with 1 level. Except for text-content (see details).
...	to ignore

**Details**

Note, text content does not get indented or put on a new line, since whites space characters are of relevance.

**Value**

character(1) vector of the formatted XML tag.

---

formattedFrame	<i>Help function to generate the formatted string for a single stack frame.</i>
----------------	---

---

### Description

Help function to generate the formatted string for a single stack frame.

### Usage

```
formattedFrame(framecall, frameN, hasSrcInfo, dirName, fileName, lineNr)
```

### Arguments

framecall	character(n): deparsed call for the given stack. Each element corresponds to a line.
frameN	integer(1): frame nummer.
hasSrcInfo	logical(1): Does the call have any source info?
dirName	character(1): the directory name of the source file.
fileName	character(1): filename of the source. Value ignored if hasSrcInfo=TRUE.
lineNr	character(1): linenr in the source where the call occurred.. Value ignored if hasSrcInfo=TRUE.

### Details

For a given frame in the stack the string is formatted as follows (substitute the arguments between the curly braces) {frameN}| {call[1]} {frameN}| {call[2]} {frameN}| {call[3]} {frameN}| {call[3]} ---> at File={dirName/fileName} Line={line}:

For example for only a single line error: 1: stop("This is a crash") ---> at File=R/my\_r\_code\_file.R Line=234

Currently all call lines are printed for a given stack. The last line with source file info only printed if hasSrcInfo=TRUE. Else it is omitted.

### Value

character(1) the formatted character string containing info of a single frame in the stacktrace



---

`getFormattedStacktrace`*Get formatted stack trace for an uncaught error from a tinytest test file.*

---

**Description**

`getFormattedStacktrace` is a helper function that formats stacktrace for uncaught errors from a tinytest run file.

**Usage**

```
getFormattedStacktrace()
```

**Details**

This function assumes that it directly called from a `withCallingHandler` error handling function! This fact is then used to remove the calling handler info from the stack such that stack directly starts from where the error was thrown.

The function also removes the calls from the stack that involve executing the `test_file`. The internals of `runTestDir` and `tinytest` are not of interest. And the highest level of the stack to consider is the top level of the `test_file`.

Note, this does mean that errors that occur on the top-level of the test file will not have a stacktrace! For example: "Error: object 'x' not found" where x is attempted to be resolved at the root levels

**Value**

`character(1)` a single length character string suitable to be printed to the end-user. In case of no stacktrace (eg the error occurred at root level of the script) `NA_character_` is returned!

---

`isSingleLengthCharNonNA`*Test if single length character non NA.*

---

**Description**

Test if single length character non NA.

**Usage**

```
isSingleLengthCharNonNA(x)
```

**Arguments**

x                    object to test.

**Value**

logical(1)

**Author(s)**

ltuijnder

---

nameTestcase

*Helper function to construct the name of a testcase*

---

**Description**

Helper function to construct the name of a testcase. Note, the character is already xml escaped.

**Usage**

nameTestcase(tinytest)

**Arguments**

tinytest          a tinytest object. (does not matter what result)

**Value**

character(1) the testcase name to use for this tinytest object.

---

passedTestcaseTag

*Construct a testcase-tag for a passed test.*

---

**Description**

Construct a testcase-tag for a passed test.

**Usage**

passedTestcaseTag(tinytest)

**Arguments**

tinytest          a tinytest object already validated to be a "PASSED" test.

**Value**

a testase XML tag

---

print.XMLtag	<i>Print method for XMLtag class.</i>
--------------	---------------------------------------

---

**Description**

Print method for XMLtag class.

**Usage**

```
## S3 method for class 'XMLtag'
print(x, ...)
```

**Arguments**

x	a XMLtag-object
...	to be ignored

**Value**

invisibly the string that was printed to stdout.

---

runTestDir	<i>Run all the test files in a directory</i>
------------	--

---

**Description**

`runTestDir()` is a drop in replacement for `tinytest::run_test_dir()` with the key difference that errors thrown from within a test file are caught and get reported with a a stacktrace in the JUnit report. In addition, some extra metrics are recored for the JUnit report, such as: timestamp, test duration, hostname and if tests are disabled (see details for more info).

**Usage**

```
runTestDir(
  dir = "inst/tinytest",
  at_home = FALSE,
  pattern = "^test.*\\. [rR]$",
  cluster = NULL,
  lc_collate = getOption("tt.collate", NA),
  ...
)
```

**Arguments**

<code>dir</code>	character(1) path to directory
<code>at_home</code>	logical(1) should local tests be run? By default FALSE. Unlike <code>tinytest::run_test_dir()</code> which is meant to be called in a local interactive context. This function is meant to be called in a non-interactive CI environment, where we want to mimic the behaviour of how tests would get run by R CMD Check. See also <code>at_home</code> documentation in <code>tinytest</code> package.
<code>pattern</code>	character(1) A regular expression that is used to find scripts in <code>dir</code> containing tests (by default <code>.R</code> or <code>.r</code> files starting with <code>test</code> ).
<code>cluster</code>	A cluster object to run the test files on. Note, it is expected that the clusters has already been prepared. Most notable, the package to test should already been loaded. <code>runTestDir</code> will load the package "tinytest" for you into the clusters. See <code>tinytest::run_test_dir()</code> for more details.
<code>lc_collate</code>	See <code>tinytest::run_test_dir()</code> .
<code>...</code>	Arguments passed on to <code>tinytest::run_test_file()</code>

**Details**

`runTestDir()` is meant as a CI-friendly alternative to the native `tinytest::run_test_dir()`. It catches errors that are raised in the tests files and adds them as a "failed" `tinytest` in the output.

`tinytest::run_test_dir()` would have let the error bubble up, stop the testing process and not report any failures from other tests. One is then also forced to look into the logs of the CI to see what the error was. The output of `runTestDir()` in combination with `writeJUnit()` will present you the error in the JUnit together with a stack trace. Next to the test results of the other files that ran without a problem.

If you prefer the behaviour of `tinytest::run_test_dir()` you can still use it in combination `writeJUnit()`.

Caught errors are returned in the output as as sub-class of `tinytest` object. This is however considered implementation detail and can be subject to change.

Note, function arguments explicitly listed in `tinytest::run_test_dir()` but not here can still still be provided via `...`

**Value**

A `tinytests2JUnit` object to be provided to the `writeJUnit()` function.

**tinytests2JUnit**

The returned object is a `tinytests2JUnit` object (note the plural). This object contains additional info compared to a `tinytests` object that is used in the JUnit report.

The following additional info will get reported:

- The timestamp per test file on when it got invoked.
- The test duration per test file.
- The system hostname per test file on where it got invoked. This is mainly of interests for different clusters.

- If a test file is disabled. A test file is considered disabled if no tests occur with in the file. It is then assumed that at the top of file some conditional statement made the test file exist early.

### See Also

- `tinytest::run_test_dir()` for how the function is intended to behave.
- `writeJUnit()` where it is expected that the output of this function to be provided to.
- `testPackage()` for an higher-level function to simply test a package.

### Examples

```
# Run tests with `tinytest`
dirWithTests <- system.file("example_tests/multiple_files",package = "tinytest2JUnit")
testresults <- runTestDir(dirWithTests)

writeJUnit(testresults) # Writes content to stdout
```

---

runTestFile	<i>Internal wrapper around <code>tinytest::run_test_file</code></i>
-------------	---

---

### Description

Internal wrapper around `tinytest::run_test_file()` that records the test duration and catches uncaught errors and logs the stacktrace of where the error occurred.

### Usage

```
runTestFile(file, ...)
```

### Arguments

file	character(1) test file to run.
...	arguments passed on to <code>tinytest::run_test_file()</code>

### Details

The response is a subclass of the `tinytests` object called: `tinytests2JUnit` object which captures additional info for the reporting to JUnit:

- Duration to run the file.
- Timestamp when the test was run.
- hostname of the computer where it was ran on.

The caught error is turned into a subclass `uncaught-error` of `tinytest`. This is implementation detail and only to be understood by `constructJUnitTag`.

If an error occurred it is captured and `uncaught-error` object (subclass of `tinytest`) is returned in the `tinytests` object. This `tinytest` object represents a "failed" tests that will get reported as an Error in the JUnit. Various aspects of the error are also captured like the the stacktrace.

**Value**

a `tinystests2JUnit` object (being a subclass of `tinystest` object).

---

`sideeffectTestcaseTag` *Construct a testcase-tag for a side-effect test.*

---

**Description**

Construct a testcase-tag for a side-effect test.

**Usage**

```
sideeffectTestcaseTag(tinystest)
```

**Arguments**

`tinystest`            a `tinystest` object already validated to be a "SIDE-EFFECT" test.

**Value**

a testase XML tag

---

<code>tag</code>	<i>XML tag</i>
------------------	----------------

---

**Description**

Create a list object that roughly mimics the behaviour of a simplistic XML tag element. Supported are XML tag-name, tag-attributes and tag-content.

**Usage**

```
tag(name, attributes = list(), content = list())
```

**Arguments**

<code>name</code>	character(1) specifying the name of the tag.
<code>attributes</code>	named-list being the XML attributes. Names = attribute names, Values = attribute value.
<code>content</code>	unnamed-list being the content XML-tag. Either child XML tag or only a single character vector being the xml text content. Currently no mixed-content is allowed. See details.

**Details**

If a character vector is in the content it is converted to a single-length character vector. See [charVecToSingleLength\(\)](#)

Mixed content eg. a text string and a child xml tag next to each other is syntactically allowed. In practice it does not occur for XML that is schema formatted with XSD (like JUnit). So for simplicity it is not supported here.

**Value**

a XML tag-object.

---

testPackage	<i>Test an R package and report the results in JUnit</i>
-------------	--

---

**Description**

Run all tests of a package and report the results as JUnit xml. This function can be seen as a drop in replacement for [tinytest::test\\_package\(\)](#) but with a key difference that uncaught errors will be caught and reported JUnit! This function is intended to be used in a test stage of a CI build.

**Usage**

```
testPackage(
  pkgname,
  file = stdout(),
  errorOnFailure = TRUE,
  testdir = "tinytest",
  lib.loc = NULL,
  at_home = FALSE,
  ncpu = NULL,
  ...
)
```

**Arguments**

pkgname	character(1). Name of the package to tests.
file	character(1)   connection: Full file path or connection object to write the JUnit xml content to. By default <code>stdout()</code> connection is used. <b>Warning</b> if the file already exist it will be overwritten!
errorOnFailure	logical(1) Should an error be raised (after writing the JUnit) when a at least one test failed? By default TRUE. This is done as a convenience to have the CI fail at the test stage on failure.
testdir	character(1) testing directory of the package. See <a href="#">?tinytest::test_package()</a> for more details.
lib.loc	character(1)   NULL Library location where the package is installed. By default: NULL meaning the package is searched on the standard <code>.libPaths()</code> .

at_home	logical(1) should local test be run? By default FALSE as we want to as closely mimic the environment of how tests would get ran in CRAN. See for more details <code>tinytest::test_package()</code> .
ncpu	positive integer(1)   cluster Either an integer specifying the amount of cpu's to parralize the testing over or a cluster object to run the tests in.
...	Extra arguments passed on to <code>runTestDir()</code>

## Details

`testPackage()` is meant as a CI-friendly alternative to the native `tinytest::test_package()`. Next to directly reporting the tests results in a JUnit xml format, it also catches errors that are raised in the tests files and reports them as "error" in the JUnit.

`tinytest::test_package()` would have let the error bubble up, stop the testing process and not report any failures from other test files. One is then also forced to look into the logs of the CI to see what the error was. `testPackage()` presents you that error in the JUnit with a stacktrace. Next to all the test results of the other files that ran without a problem.

If you prefer the behaviour from `tinytest::test_package()`, you can still use it in combination with `writeJUnit()` if all tests results pass.

Just like `tinytest::test_package()` an error is raised if at least one failure occured during testing. Obviously caught errors are also seen as failures. This error is raised **after** the test results have been written away to the file, such that your CI can still pick it up and report the failure. The error raising is done as a convenience to stop the CI from continue if test-failure occured. You can opt-out of this behaviour by setting the `errorOnFailure` parameter to FALSE. Then a case `tinytests2JUnit` object is returned (a sub-class of `tinytests` object containing addition info for the JUnit). Caught errors are also captured in this object as `tinytest-objects`. They actually have a special sub-class but this is considered an internal implementation detail.

`testPackage()` is NOT meant to be called from within your tests/tinytests.R file! Tests invoked by R CMD Check or on CRAN should still make use of `tinytest::test_package()`. This function is only meant to be called from within a testing step in your CI to report the test results in an JUnit xml format.

## Value

If `errorOnFailure = FALSE`, a `tinytests2JUnit` object (a subclass of `tinytests` object that captures more info for export to JUnit). Else, an error is raised if at least on failure occurs. Meant as convenience to automatically stop the CI build.

## Side-effects

Side effects are registered as 'passed' tests in the JUnit output and have been given a status "SIDE-EFFECT". The call and diff is also returned in the standard-output of the testcase tag.

They are not considred failures and would thus not stop a pipeline.

## tinytests to JUnit

To comply the the JUnit specification the tests results are adapted as follows:

- A single test run `tinytests` is mapped to a `<testsuites>` tag.



- All `tinytest` results from a single file are mapped to a single `<testsuite>` tag.
  - The name of the testsuite is equal to the test file name (without the file suffix)
- An individual `tinytest` object (eg. a single `expect_*` exception test) is mapped to a `<testcase>` tag.
  - The name of the testcase is equal to the `fileName` + Line specification of where the expect statement is performed + the info.

For reference: <https://llg.cubic.org/docs/junit/>

### See Also

`runTestDir()` and `tinytest::test_package()`.

### Examples

```
tmpFile <- tempfile(fileext = ".xml")
testPackage("tinytest", file = tmpFile, verbose = 0)
```

---

writeJUnit

*Write the results of a tinytests-object into JUnit xml report.*

---

### Description

Write the `tinytests`-object to a JUnit XML reporting file. If a `tinytests2JUnit` is provided (returned by `runTestDir()`) more info will get reported.

### Usage

```
writeJUnit(tinytests, file = stdout(), overwrite = TRUE)
```

### Arguments

<code>tinytests</code>	<code>tinytests</code> -object to convert to JUnit xml.
<code>file</code>	<code>character(1)</code>   connection: Full file path or connection object to write the JUnit xml content to. By default <code>stdout()</code> connection is used.
<code>overwrite</code>	<code>logical(1)</code> : should the file be overwritten if it already exist? By default TRUE.

### Value

`invisible(TRUE)` Might get another use in the future.

### Errors

In case of `overwrite = FALSE` and the file already exists an error is thrown.

**Side-effects**

Side effects are registered as 'passed' tests in the JUnit output and have been given a status "SIDE-EFFECT". The call and diff is also returned in the standard-output of the testcase tag.

They are not considered failures and would thus not stop a pipeline.

**tinytests to JUnit**

To comply the the JUnit specification the tests results are adapted as follows:

- A single test run `tinytests` is mapped to a `<testsuites>` tag.
- All `tinytest` results from a single file are mapped to a single `<testsuite>` tag.
  - The name of the testsuite is equal to the test file name (without the file suffix)
- An individual `tinytest` object (eg. a single `expect_*` exception test) is mapped to a `<testcase>` tag.
  - The name of the testcase is equal to the `fileName` + Line specification of where the expect statement is performed + the info.

For reference: <https://llg.cubic.org/docs/junit/>

**See Also**

The JUnit XML report format: <https://llg.cubic.org/docs/junit/>

**Examples**

```
# Run tests with `tinytest`
dirWithTests <- system.file("example_tests/multiple_files",package = "tinytest2JUnit")
testresults <- runTestDir(dirWithTests)

writeJUnit(testresults) # Writes content to stdout

tmpFile <- tempfile(fileext = ".xml")
writeJUnit(tinytests = testresults, file = tmpFile)
```

---

[.tinytests2JUnit      *tinytestJUnit test results*

---

**Description**

An object of class `tinytests2JUnit`. Note the plural. A subclass of `tinytest::tinytests()` containing extra info recordings that are used in the export to JUnit.

**Usage**

```
## S3 method for class 'tinytests2JUnit'
x[i]
```

### Arguments

- x                    tinytests2JUnit object to subset.
- i                    object to subset the x with.

### Details

Following details are recorded when running the tests files and stored as additional attributes to the object:

- **fileDurations:** named-numeric(n). Names = filename of tests files, value = duration in seconds on how long the test file took to run.
- **fileTimestamps:** named-character(n). Names = filename of tests files, value = timestamp when the test was invoked.
- **fileHostnames:** named-character(n). Names = filename of tests files, value = The hostname of the system that ran the tests. (Usefull in combination with clusters).
- **disabled:** character. A character vector of filenames where no tests were ran. They are flagged as disabled tests.

# Index

- [.tinytests2JUnit, 18
- charVecToSingleLength, 2
- charVecToSingleLength(), 15
- classnameTestcase, 3
- constructFailureDescription, 3
- constructTestcaseTag, 4
- constructTestsuitesTag, 4
- constructTestsuiteTag, 5
- errorTestcaseTag, 5
- escapeXml, 6
- escapeXmlText, 6
- failureTestcaseTag, 7
- format.XMLtag, 7
- formattedFrame, 8
- getFormattedStacktrace, 9
- isSingleLengthCharNonNA, 9
- nameTestcase, 10
- passedTestcaseTag, 10
- print.XMLtag, 11
- runTestDir, 11
- runTestDir(), 4, 11, 12, 16, 17
- runTestFile, 13
- sideeffectTestcaseTag, 14
- tag, 14
- testPackage, 15
- testPackage(), 13, 16
- tinytest::run\_test\_dir(), 4, 11–13
- tinytest::run\_test\_file(), 12, 13
- tinytest::test\_package(), 4, 15–17
- tinytest::tinytests(), 18
- tinytests2JUnit ([.tinytests2JUnit), 18
- writeJUnit, 17
- writeJUnit(), 12, 13, 16