Package 'wyz.code.offensiveProgramming'

September 21, 2019

Type Package

Title Wizardry Code Offensive Programming

Version 1.1.11

Author Fabien Gelineau <neonira@gmail.com>

Maintainer Fabien Gelineau <neonira@gmail.com>

Description Allows to change R coding from defensive programming (i.e. many input parameter checks implementation required) to offensive programming (none/reduced number of parameter checks required). Provides code instrumentation to ease this change. Should reduce the code size as many controls and type checks have no more reason to exist. Should also speed up processing as many checks will be reduced to single check.

Encoding UTF-8

LazyData true

License GPL-3

Depends R (>= 3.5)

Imports methods, data.table (>= 1.11.8), tidyr, lubridate (>= 1.7.4), stringr (>= 1.4.0), R6 (>= 2.4.0)

Suggests testthat, knitr, rmarkdown

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2019-09-21 11:50:02 UTC

R topics documented:

`defineEvaluationModes` *Define evaluation modes*

Description

Get all predefined evaluation mode names

Usage

```
defineEvaluationModes()
```

Value

A vector of strings, each representing a reusable evaluation mode name.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to EvaluationMode.

Examples

```
##---- typical case ----
defineEvaluationModes()
```

`defineFunctionReturnTypesParameterName`
define function return type parameter name

Description

Provides the parameter name to use to define function return type.

Usage

```
defineFunctionReturnTypesParameterName()
```

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to EvaluationMode.

Examples

```
##---- typical case ----
defineFunctionReturnTypesParameterName()
```

defineTestCaseDefinitionsParameterName *Test case definition parameter name*

Description

Define the paramater name to hold test case definitions

Usage

```
defineTestCaseDefinitionsParameterName()
```

Value

A single string that is the parameter name to use.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to EvaluationMode. See sibling EvaluationMode.

Examples

```
##---- typical case ----
defineTestCaseDefinitionsParameterName()
```

EvaluationMode *Evaluation mode definition*

Description

Class to define your evaluation mode

Usage

```
EvaluationMode(value_s_1 = defineEvaluationModes()[2])
```

Arguments

value_s_1	one string that must come from defineEvaluationModes

Value

An object that is an R `environment`.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to `defineEvaluationModes`

Examples

```
##---- typical case ----
EvaluationMode(defineEvaluationModes()[3])
```

`FunctionParameterName` *Function parameter name*

Description

Class to define and handle a function parameter

Usage

```
FunctionParameterName(name_s_1)
```

Arguments

`name_s_1`	the name of the parameter

Details

The name of the parameter should be a semantic name. A semantic name is a compound string based on a special format allowing to distinguish by the name, the parameter type, and to express some length constraints.

Value

An object that is an R `environment`. Use functions `isSemanticName`, `isPolymorphic`, `isEllipsis`, `isValid` to check the provided name. Functions `get*` allows to retrieve parts of the name.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to defineEvaluationModes

Examples

```
##---- typical case ----
fpn <- FunctionParameterName('values_s_7m')
fpn$isPolymorphic()
fpn$isSemanticName()
fpn$isValid()
fpn$getTypeSuffix() # 's'
fpn$getLengthSpecification() # '7m'
fpn$getLengthSuffix() # 7
fpn$getLengthModifier() # 'm'

fpn <- FunctionParameterName('object_')
fpn$isPolymorphic()
fpn$isSemanticName()
fpn$isValid()
```

FunctionParameterTypeFactory

Function parameter type factory

Description

This factory is a parameter type check factory. It provides type checking for each allowed type.

Usage

```
FunctionParameterTypeFactory()
```

Details

Many common types are already recorded and available through the factory. Use the function getRecordedTypes to get more insight.

If you desire to verify a type instrumentation, just use checkSuffix function. If you want to add an instrumentation for a new type, use addSuffix function.

See examples below for more hands-on approach.

Value

An object that is an R environment.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to defineEvaluationModes

Examples

```
##---- typical case ----
ff <- FunctionParameterTypeFactory()
ff$checkSuffix('b') # TRUE

# see verify_function recorded for 'boolean' entries
ff$getRecordedTypes()[suffix == 'b']$verify_function[[1]]

# record a new entry for suffix 'wo'
ff$addSuffix('wo', "wo class", function(o_) is(o, "wo")) # TRUE
ff$getRecordedTypes()[suffix == 'wo']
```

`getEllipsisName` *Get ellipsis.*

Description

Use method `getEllipsisName` to get ellipsis argument value.

Usage

```
getEllipsisName()
```

Value

A string with value `...`, no more no less.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# typical test
getEllipsisName()
#[1] "..."
```

`getObjectClassKind` *Get* R *object class kind*

Description

Get the class kind of an R object as a string.

Usage

```
getObjectClassKind(object_o_1)
```

Arguments

`object_o_1`	the object to analyze. See `is.object`.

Value

A single `character` value, taken in set `"S3"`, `"S4"`, `"RC"`, `"R6"`, `"environment"`, `"unknown"`.

When provided `object_` is not an R object, then value `NA_character_` is returned.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
##---- typical case ----
getObjectClassKind(new.env())
# [1] NA

myrc <- setRefClass("RC",
  fields = list(x = "numeric"),
  methods = list(
    initialize = function(x = 1) .self$x <- x,
    getx = function() x,
    inc = function(n = 1) x <<- x + n
  )
)

getObjectClassKind(myrc$new())
# [1] RC

myr6 <- R6::R6Class("R6",
  public = list(
    x = NULL,
    initialize = function(x = 1) self$x <- x,
    getx = function() self$x,
    inc = function(n = 1) self$x <- x + n
```

```
  )
)

getObjectClassKind(myr6$new())
# [1] R6
```

`getObjectClassNames` *Retrieve Function Arguments.*

Description

Use method `getObjectClassNames` to get the class names of an object (see `is.object`).)

Usage

```
getObjectClassNames(object_o_1)
hasMainClass(object_o_1, classname_s_1)
```

Arguments

`object_o_1`	the object to analyze.
`classname_s_1`	the class to match the `classname` entry of the returned value of `getObjectClassNames)`.

Value

A `list` with two character entries. First one is named `classname`, provides the main classname (the one found in first position). Second one is named `classnames`, provides all the class names born by the `object_`.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# typical test
getObjectClassNames(getObjectClassNames(factor(letters[1:3])))
#$classname
#[1] "factor"

#$classnames
#[1] "factor"

# another test
getObjectClassNames(new.env())
#$classname
#[1] NA
```

```
#$classnames
#[1] "environment"
```

`getObjectFunctionArgumentNames` *Retrieve Function Arguments.*

Description

Use method `getObjectFunctionArgumentNames` to get the function argument names of an object (see `is.object`).

Usage

```
getObjectFunctionArgumentNames(object_o_1, allNames_b_1 = TRUE)
```

Arguments

`object_o_1`	the object to analyze.
`allNames_b_1`	A boolean value. Passed to function `getObjectFunctionNames` to restrict output if needed.

Value

A `list`. Entries are named with function names. Each entry is of type character, and holds function argument names. Could be empty if function takes no argument.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# typical test
MyEnv <- function() {
     self <- environment()
     class(self) <- append('MyEnv', class(self))
     f <- function(x_3, y_3n) x_3 + y_3n
     self
}

getObjectFunctionArgumentNames(MyEnv())
#$f
#[1] "x_3" "y_3n"
```

getObjectFunctionNames

Retrieve Function Names From Object

Description

Use method getObjectFunctionNames to get the function names of an object (see is.object).)

Usage

```
getObjectFunctionNames(object_o_1, allNames_b_1 = FALSE)

getClassTypicalFunctionNames(object_o_1)
```

Arguments

object_o_1	the object to analyze.
allNames_b_1	A boolean value. When TRUE, uses getClassTypicalFunctionNames to restrict the set of function names returned.

Details

Function getClassTypicalFunctionNames gives back function names that are related to R class style, and automatically added by R to your class object.

Value

A vector of function names (character).

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# typical test
MyEnv <- function() {
     self <- environment()
     class(self) <- append('MyEnv', class(self))
     f <- function(x_3, y_3n) x_3 + y_3n
     self
}

getObjectFunctionNames(MyEnv())
# [1] "f"

# another test
```

```
getObjectFunctionNames(new.env())
#[1] NA
```

`isAuditable` *Is auditable.*

Description

Use method `isAuditable` to know if code is auditable

Usage

```
isAuditable()
```

Value

A boolean value. To turn value to `TRUE`, environment variable `OP_AUDITABLE` must be set to a value that does not match an empty string.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# typical case
isAuditable()
# FALSE
```

`packageFunctionsInformation`
Package functions information

Description

A reminder of available functions from this package, and, most common usage intent. A poor man CLI cheat sheet.

Usage

```
packageFunctionsInformation()
```

Value

A `data.table` with following column names

`function`	a function name provided by this package
`elaboration`	a boolean stating if function could/should be use for elaboration purpose
`verification`	a boolean stating if function could/should be use for verification purpose
`exploitation`	a boolean stating if function could/should be use for exploitation purpose
`information`	a boolean stating if function could/should be use for informative purpose
`kind`	the classification kind of the chosen function
`user`	a typical user role/profile that may use the function

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer also to package vignettes.

Examples

```
##---- typical case ----
packageFunctionsInformation()
```

`print.EvaluationMode` *Print generic method for S3 class* `EvaluationMode`

Description

Prints the `EvaluationMode` data

Usage

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

Arguments

`x`	the `EvaluationMode` object to consider
`...`	any other argument, passed to print.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
b <- EvaluationMode(defineEvaluationModes()[2])
print(b)
```

---

print.TestCaseDefinition *Print generic method for S3 class* TestCaseDefinition

---

Description

Prints the TestCaseDefinition data

Usage

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

Arguments

x	the TestCaseDefinition object to consider
...	any other argument, passed to print.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
b <- TestCaseDefinition(list(1L, 2L), 3L, 'sum of 2 integers')
print(b)
```

---

retrieveFactory *Retrieve factory*

Description

As factory may be modified, this function allows you to make changes and to record them in a global environment variable, to ease reuse.

Usage

```
retrieveFactory(functionParameterTypeFactory_o_1 = NULL)
```

Arguments

functionParameterTypeFactory_o_1
: The factory you want to use. Will provide the standard factory by default.

Details

Sets typeFactory in .GlobalEnvironment in order to reuse it transparently wherever required

Value

An object that is an R FunctionParameterTypeFactory.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to FunctionParameterTypeFactory

Examples

```
##---- typical case ----
ff <- FunctionParameterTypeFactory()
ff$addSuffix('wo', "wo class", function(o_) is(o_, "wo"))
ff$addSuffix('yo', "yo class", function(o_) is(o_, "yo"))
ff$addSuffix('zo', "zo class", function(o_) is(o_, "zo"))
retrieveFactory(ff) # set and retrieves the factory
```

---

retrieveFunctionArgumentNames

Retrieve Function Argument Names.

Description

Use method retrieveFunctionArgumentNames to get names of function or primitive arguments.

Usage

```
retrieveFunctionArgumentNames(fun_f_1)
```

Arguments

fun_f_1
: a function or primitive. Not a string!

Value

A vector. See formalArgs.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# typical test on a primitive
retrieveFunctionArgumentNames(sin)
#[1] "x"

# typical test on a function
retrieveFunctionArguments(ls)
#[1] "name"      "pos"       "envir"     "all.names" "pattern"   "sorted"
```

---

retrieveFunctionArguments

Retrieve Function Arguments.

Description

Use method `retrieveFunctionArguments` to get function or primitive arguments.

Usage

```
retrieveFunctionArguments(fun_f_1)
```

Arguments

`fun_f_1`	a function or primitive. Not a string!

Value

A pairlist. See `formals`.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# typical test on a primitive
retrieveFunctionArguments(sin)
#$x
#

# typical test on a function
retrieveFunctionArguments(ls)
```

```
#$name


#$pos
#-1L

#$envir
#as.environment(pos)

#$all.names
#[1] FALSE

#$pattern
#

#$sorted
#[1] TRUE
```

`retrieveFunctionReturnTypes` *Retrieve function return types*

Description

From an instrumented class, retrieve the function return types definition.

Usage

```
retrieveFunctionReturnTypes(object_o_1,
functionParameterTypeFactory_o_1 = retrieveFactory())
```

Arguments

`object_o_1`	the object to consider
`functionParameterTypeFactory_o_1`	the function parameter type factory to consider

Value

A polymorphic return that is either

a `list`	as returned by the `verifyObjectNames` function
another `list`	as returned by the `verifyFunctionReturnTypesDefinition` function
a `data table`	the function parameter types definition as declared in the source class

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to `defineEvaluationModes`

Examples

```
##---- typical case ----
library('data.table')
source(system.file('code-samples/no-defs/Addition.R',
                   package = 'wyz.code.offensiveProgramming'))
source(system.file('code-samples/frt-defs/good/full/AdditionFI.R',
                   package = 'wyz.code.offensiveProgramming'))
retrieveFunctionReturnTypes(AdditionFI()) # works, renders a data.table
retrieveFunctionReturnTypes(Addition()) # fails, renders a list
```

`retrieveTestCaseDefinitions` *Retrieve test cases definitions*

Description

From an instrumented class, retrieve the test case definitions.

Usage

```
retrieveTestCaseDefinitions(object_o_1,
   functionParameterTypeFactory_o_1 = retrieveFactory())
```

Arguments

`object_o_1`	the object to consider
`functionParameterTypeFactory_o_1`	the function parameter type factory to consider

Value

A polymorphic return that is either

a `list`	as returned by the `verifyObjectNames` function
another `list`	as returned by the `verifyFunctionReturnTypesDefinition` function
a `data table`	the test case definitions as declared in the source class

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to defineEvaluationModes

Examples

```
##---- typical case ----
library('data.table')
source(system.file('code-samples/tcd-defs/good/partial/AdditionTCPartial.R',
                   package = 'wyz.code.offensiveProgramming'))
source(system.file('code-samples/no-defs/Addition.R',
                   package = 'wyz.code.offensiveProgramming'))
retrieveTestCaseDefinitions(AdditionTCPartial()) # works, renders a data.table
retrieveTestCaseDefinitions(Addition()) # fails, renders a list
```

---

runFunction *Run a function*

Description

Run a function from an object, according to the mentioned evaluation mode, and to the chosen type factory

Usage

```
runFunction(object_o_1, functionName_s_1, arguments_l,
evaluationMode_o_1, functionParameterTypeFactory_o_1 = retrieveFactory())
```

Arguments

object_o_1	the object to consider
functionName_s_1	a single string that is the name of the function to run
arguments_l	a list of arguments to pass to the function
evaluationMode_o_1	an evaluation mode object. See EvaluationMode
functionParameterTypeFactory_o_1	A type factory. See retrieveFactory

Value

A `list` with names

`status`	a single boolean. Always `TRUE` when evaluation mode is `"standard_R_evaluation"`. Otherwise, will reflect result validity in the chose evaluation mode.
`value`	the result of the computation, might be a scalar or not, a warning, an error, ...
`mode`	the evaluation mode used to check the results
`function_return_type_check`	available if mode is different of `"standard_R_evaluation"`
`parameter_type_checks`	available if mode is `"type_checking_inforcement"`

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to `FunctionParameterTypeFactory` and `runFunction`.

Examples

```
##---- typical case ----
library('data.table')
source(system.file('code-samples/frt-defs/good/full/AdditionFI.R',
                   package = 'wyz.code.offensiveProgramming'))
fi <- AdditionFI()
runFunction(fi, 'addDouble', list(34, 44.6), EvaluationMode(defineEvaluationModes()[1]))
runFunction(fi, 'addDouble', list(34, 44.6), EvaluationMode(defineEvaluationModes()[2]))
runFunction(fi, 'addDouble', list(34, 44.6), EvaluationMode(defineEvaluationModes()[3]))
```

`runTestCase` *Run test cases*

Description

Run specified test cases under the given evaluation mode

Usage

```
runTestCase(object_o_1, testCaseIndexes_i, evaluationMode_o_1 = EvaluationMode())
```

Arguments

object_o_1	The R object to consider
testCaseIndexes_i	a vector of numbers identifying the test cases to run
evaluationMode_o_1	the evaluation mode to use. see EvaluationMode

Value

A list with two names

raw	a list with one entry for each test ran, holding all data and metadata related to the test
synthesis	a summary data.table that allows to see at a glance all the tests results. Also eases comparisons of results between various evaluation modes.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to defineEvaluationModes

Examples

```
##---- typical case ----
library('data.table')
source(system.file('code-samples/both-defs/good/full/AdditionTCFI_G1.R',
                   package = 'wyz.code.offensiveProgramming'))
em <- EvaluationMode('type_checking_enforcement')
runTestCase(AdditionTCFI_G1(), c(3, 5, 7), em)
```

---

runTransientFunction *Run transient function*

Description

Run a function in a transient (non persistent) context.

Usage

```
runTransientFunction(function_f_1, arguments_l, evaluationMode_o_1,
function_return_type_s_1, functionParameterTypeFactory_o_1 = retrieveFactory())
```

Arguments

function_f_1	a single R function
arguments_l	a list of arguments to pass to the function
evaluationMode_o_1	an evaluation mode object. See EvaluationMode
function_return_type_s_1	a semantic parameter name, given as a string, to express expected function return type
functionParameterTypeFactory_o_1	A type factory. See retrieveFactory

Value

A list with names

status	a single boolean. Always TRUE when evaluation mode is "standard_R_evaluation". Otherwise, will reflect result validity in the chose evaluation mode.
value	the result of the computation, might be a scalar or not, a warning, an error, ...
mode	the evaluation mode used to check the results
function_return_type_check	available if mode is different of "standard_R_evaluation"
parameter_type_checks	available if mode is "type_checking_inforcement"

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to runFunction

Examples

```
##---- typical case ----
em <- EvaluationMode(defineEvaluationModes()[3])
h <- function(x_s) x_s
runTransientFunction(h, list('neonira'), em, 'x_s')
runTransientFunction(h, list(pi), em, 'x_s')
runTransientFunction(h, list(pi), em, 'x_d')
```

`TestCaseDefinition` *Test case definition*

Description

Defines a test case

Usage

```
TestCaseDefinition(params_l, expectedResult_, description_s_1)
```

Arguments

`params_l`	a list that holds the test case input values
`expectedResult_`	test case expected result. This will be used to compare with function execution results
`description_s_1`	a single string, test case description

Details

Test case definition takes sense only when intimately correlated with a function

Value

An object that is an R `environment`.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to `FunctionParameterTypeFactory`

Examples

```
##---- typical case ----
tcd <- TestCaseDefinition(list(1:5), 15, 'sum of 5 first non nul integers')
tcd <- TestCaseDefinition(list(1:7, 3:5, sample(1:100, 19, FALSE)),
                          list(3:5), 'extract smallest length from input')
```

`verifyClassName` *Verify class name*

Description

Class name must comply with a policy. This function allows to check compliance.

Usage

```
verifyClassName(name_s = "MyClassName", strictSyntax_b_1 = TRUE)
```

Arguments

`name_s`	The class name to be checked
`strictSyntax_b_1`	A boolean value. When `TRUE`, allowed character set is `[A-Za-z0-9]+`. A class name must start with an uppercase letter. The name is required to be camel cased, although this cannot be checked. When `FALSE`, allowed character set is `[A-Za-z0-9_.]+`. Classic R class naming applies.

Value

A boolean value, either `TRUE` or `FALSE`

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to `defineEvaluationModes`

Examples

```
##---- typical case ----
verifyClassName('matrix')
verifyClassName('matrix', FALSE)
```

verifyFunctionName *Verify function name*

Description

Function name must comply with a policy. This function allows to check compliance.

Usage

```
verifyFunctionName(name_s = "aSimpleFunctionName", strictSyntax_b_1 = TRUE)
```

Arguments

name_s	The function name to be checked
strictSyntax_b_1	A boolean value. When TRUE, allowed character set is [A-Za-z0-9]+. A function name must start with a lowercase letter. The name is required to be camel cased, although this cannot be checked. When FALSE, allowed character set is [A-Za-z0-9_.]+. Classic R function naming applies.

Value

A boolean value, either TRUE or FALSE

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to defineEvaluationModes

Examples

```
##---- typical case ----
verifyFunctionName('matrix')
verifyFunctionName('matrix', FALSE)
```

`verifyFunctionReturnTypesDefinition` *Verify function return types definition*

Description

Function return types definition has to comply with many rules. This functions checks for this compliances and helps in dectection of uncompliances.

Usage

```
verifyFunctionReturnTypesDefinition(object_o_1,
requiresFullInstrumentation_b_1 = TRUE,
functionParameterTypeFactory_o_1 = retrieveFactory())
```

Arguments

`object_o_1`	The object to be considered
`requiresFullInstrumentation_b_1`	a boolean stating if full instrumentation is required
`functionParameterTypeFactory_o_1`	a type factory to check parameter values and types

Details

When `requiresFullInstrumentation_b_1` is `TRUE`, each function must have an entry in the test case parameter definition.

Value

A `list` with names

`validity`	a single boolean value
`class`	the class name of the provided object
`intent`	the stage of the failure, provides hint about the faced issue
`message`	some hints to resolve the issue(s).

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to `defineTestCaseDefinitionsParameterName`.

Examples

```
##---- typical case ----
library('data.table')
source(system.file('code-samples/frt-defs/good/full/AdditionFI.R',
                   package = 'wyz.code.offensiveProgramming'))
fi <- AdditionFI()
print(verifyFunctionReturnTypesDefinition(fi))
print(verifyFunctionReturnTypesDefinition(fi, FALSE))
```

`verifyObjectNames` *Verify object names*

Description

Verify object class name, object function names, and object function parameter names, and provides a synthesis of checks.

Proceeds also to some introspection on object to identify instrumentation of function return types and test case definitions. Provides information about completeness of instruction, and about missing functions and test cases.

Usage

```
verifyObjectNames(object_o_1, functionParameterTypeFactory_o_1 = retrieveFactory())
```

Arguments

`object_o_1`	the object to be checked
`functionParameterTypeFactory_o_1`	the type factory to check parameter types

Value

A `list` with following names

`class_name`	the class name of the provided object
`supports_strict_compliance`	a single boolean
`supports_lazy_compliance`	a single boolean
`class_name_compliance`	a boolean value expression class name compliance
`class_name_compliance`	a vector of booleans, where names are the function names and values express the name compliance
`class_name_compliance`	a `data.table` exposing the name compliance and the semanting name compliance for each paramter

owns_function_return_type_information
: a single boolean

can_be_type_checked
: a single boolean

is_function_fully_instrumented
: a single boolean

missing_function
: a vector of uninstrumented function names

owns_test_case_definitions
: a single boolean

is_test_case_fully_instrumented
: a single boolean

missing_test_cases
: a single boolean

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to verifyClassNameand verifyFunctionName.

Examples

```
##---- typical case ----
library('data.table')
source(system.file('code-samples/frt-defs/good/full/AdditionFI.R',
                   package = 'wyz.code.offensiveProgramming'))
fi <- AdditionFI()
print(verifyObjectNames(fi))
```

---

verifyTestCaseDefinitions

Verify test case definitions

Description

Test case definitions has to comply with many rules. This functions checks for this compliances and helps in dectection of uncompliances.

Usage

```
verifyTestCaseDefinitions(object_o_1, requiresFullInstrumentation_b_1 = TRUE,
functionParameterTypeFactory_o_1 = retrieveFactory())
```

Arguments

`object_o_1`	The object to be considered
`requiresFullInstrumentation_b_1`	a boolean stating if full instrumentation is required
`functionParameterTypeFactory_o_1`	a type factory to check parameter values and types

Details

When `requiresFullInstrumentation_b_1` is `TRUE`, each function must have an entry in the test case parameter definition.

Value

A `list` with names

`validity`	a single boolean value
`class`	the class name of the provided object
`intent`	the stage of the failure, provides hint about the faced issue
`message`	some hints to resolve the issue(s).

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to `defineTestCaseDefinitionsParameterName`.

Examples

```
##---- typical case ----
library('data.table')
source(system.file('code-samples/tcd-defs/good/full/AdditionTC.R',
                   package = 'wyz.code.offensiveProgramming'))
tc <- AdditionTC()
print(verifyTestCaseDefinitions(tc))
```

Index