Package 'wyz.code.offensiveProgramming'

September 25, 2023

Type Package

Title Wizardry Code Offensive Programming

Version 1.1.24

Author Fabien Gelineau <neonira@gmail.com>

Maintainer Fabien Gelineau <neonira@gmail.com>

Description Allows to turn standard R code into offensive programming code.
Provides code instrumentation to ease this change and tools to assist and accelerate code production and tuning while using offensive programming code technics.
Should improve code robustness and quality. Function calls can be easily verified on-demand or in batch mode to assess parameter types and length conformities.
Should improve coders productivity as offensive programming reduces the code size due to reduced number of controls all along the call chain.
Should speed up processing as many checks will be reduced to one single check.

Encoding UTF-8

License GPL-3

Depends R (>= 4.0)

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

Suggests testthat, knitr, rmarkdown

RoxygenNote 6.1.1

VignetteBuilder knitr

URL https://neonira.github.io/offensiveProgrammingBook_v1.2.2/

NeedsCompilation no

Repository CRAN

Date/Publication 2023-09-25 07:20:02 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])
```

exploreObjectNamesVerification *Verify object names*

Description

Human readable output synthetized from verifyObjectNames

Usage

```
exploreObjectNamesVerification(object_o_1,
                        what_s_1 = c("names", "return type", "test cases", "*")[1])
```

Arguments

object_o_1	the object to be checked
what_s_1	a single string that expresses what you want to focus on, should start by one of [nrt* character. Star means show all.

Value

The same value as verifyObjectNames is returned in invisible mode.

It adds stdout output to give very short synthesis about object names and content.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to verifyClassName and verifyFunctionName.

Examples

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

---

findFilesInPackage *find files in package*

Description

Use function findFilesInPackage to find files in package.

Usage

```
findFilesInPackage(filenames_s, packageName_s_1)
```

Arguments

filenames_s	An unconstrained vector of string values.
packageName_s_1	A length-1 vector of string values.

Value

This function is vectorized. It returns a list with one entry for each file searched for.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# ------- example 1 -------
findFilesInPackage(c("AdditionTCFIG1.R", "Addition_TCFI_Partial_R6.R",
                     "Addition_TCFI_Partial_S3.R"),
                   "wyz.code.offensiveProgramming")
# .../wyz.code.offensiveProgramming/code-samples/both-defs/good/full/AdditionTCFIG1.R
# .../wyz.code.offensiveProgramming/code-samples/both-defs/good/partial/Addition_TCFI_Partial_R6.R
# .../wyz.code.offensiveProgramming/code-samples/both-defs/good/partial/Addition_TCFI_Partial_S3.R

# ------- example 2 -------
findFilesInPackage("datatable-intro.html", "data.table")
# .../data.table/doc/datatable-intro.html
```

FunctionParameterName *Function parameter name*

Description

Class to define and handle a function parameter

Usage

```
FunctionParameterName(name_s_1)
```

Arguments

`name_s_1`	a `string` that is 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

Get ellipsis argument name 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

Retrieve 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`	a `string` that is the class name to match the `classname` entry returned by `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

Retrieve function argument names from an 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>

See Also

See `is.object`.

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

Retrieve 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
```

`identifyOPInstrumentationLevel` *Identify Offensive Programming Instrumentation Level*

Description

Provide short information about offensive programming instrumentation level

Usage

```
identifyOPInstrumentationLevel(object_o_1 = NULL,
                               methodName_s_1 = NA_character_)
```

Arguments

`object_o_1`	the `object` to be checked
`methodName_s_1`	the `function` name to consider, if any.

Value

A `list` with following names

`offensive_programming`
: a single `boolean`

`full_instrumentation`
: a single `boolean`

`semantic_naming`
: a single `boolean`

`function_return_type`
: a single `boolean`

`test_case_definition`
: a single `boolean`

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer to verifyClassName and verifyFunctionName.

Examples

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

#$offensive_programming
#[1] TRUE

#$full_instrumentation
#[1] FALSE

#$semantic_naming
#[1] TRUE

#$function_return_type
#[1] TRUE

#$test_case_definition
#[1] FALSE
```

---

isAuditable *Is Auditable*

Description

Retrieve option telling if code is auditable

Usage

```
isAuditable()
```

Value

A boolean value. To turn value to TRUE, set option op_audit to TRUE.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

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

`matchFunctionSignature` *Retrieve Function Arguments.*

Description

Compare two functions signatures and tells if they are exactly the same.

Usage

```
matchFunctionSignature(aFunction_f_1, aFunctionTemplate_f_1 = function(){})
```

Arguments

`aFunction_f_1`	a `function` or `primitive`. Not a string!
`aFunctionTemplate_f_1`	a `function` or `primitive` to be used as model. Not a string!

Details

To get `TRUE` as result, function and function model must share exactly the sames attributes names and values, including default values if any used.

Value

A `boolean` value.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
matchFunctionSignature(sum, function(..., na.rm = FALSE) { NULL })
# [1] TRUE

matchFunctionSignature(sum, function(..., na.rm) { NULL })
#[1] FALSE
```

opInformation *Package functions information*

Description

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

Usage

```
opInformation()
```

Value

A data.table with following columns

name	the object name
category	the category of the object describe by function name. Could be CLASS, FUNCTION or DATA.
nature	either INTERNAL or EXPORTED.
stratum	the stratum the object belongs to. Values are CORE, LAYER_1, LAYER_2, LAYER_3.
phasing	main usage phase of the object. Values are DESIGN, BUILD, TEST, RUN, MAINTAIN, EVOLVE and TRANSVERSAL.
intent	main global intent of the object. Values are PARTS_BUILDING, PARTS_ASSEMBLY, QUALITY_CONTROL, FEEDBACK, STATISTICS, CONTENT_GENERATION and UTILITIES.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

Refer also to package vignettes.

Examples

```
##---- typical case ----
opInformation()
```

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.FunctionParameterName *Print generic method for S3 class* FunctionParameterName

Description

Prints the FunctionParameterName data

Usage

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

Arguments

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

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
fn <- FunctionParameterName('x_s')
print(fn)
```

---

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 the type factory object*

Description

As factory may be modified, this function allows you to make changes and to record them in your own specialized type factory, to match various needs and ease reuse.

Usage

```
retrieveFactory()
```

Details

Retrieves a `FunctionParameterTypeFactory` from options variable named `op_type_factory` or provides a default type factory.

Value

An R `object` that is a `FunctionParameterTypeFactory`.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
##---- typical case ----
ff <- retrieveFactory()
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"))

options('op_type_factory' = ff)
fg <- retrieveFactory() # retrieves the factory pointed by R variable ff
fg$getRecordedTypes()[suffix %in% c('wo', 'yo', 'zo')] # right behavior !

# wrong behavior as retrieveFactory will provide the default factory and not yours!
options('op_type_factory' = ff)
fh <- retrieveFactory() # retrieves the default factory
fh$getRecordedTypes()[suffix %in% c('wo', 'yo', 'zo')]
```

retrieveFunctionArgumentNames
Retrieve Function Argument Names.

Description

Retrieve function argument names from a function or a primitive.

Usage

```
retrieveFunctionArgumentNames(fun_f_1)
```

Arguments

fun_f_1	a function or primitive. Not a string!

Value

A vector of strings that are the function names.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

See retrieveFunctionArguments. See formalArgs.

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

Retrieve function arguments to get arguments from a function or a primitive.

Usage

```
retrieveFunctionArguments(fun_f_1)
```

Arguments

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

Value

A `pairlist`.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

See Also

See `retrieveFunctionArguments`. See `formalArgs`.

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

Retrieve the function return type definitions from an `object`.

Usage

```
retrieveFunctionReturnTypes(object_o_1)
```

Arguments

`object_o_1`	the `object` 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
```

`retrievePackageFunctionNames`
Retrieve Package Function Names

Description

Get the function names from a package name

Usage

```
retrievePackageFunctionNames(packageName_s_1, libraryPath_s_1 = .libPaths()[1])
```

Arguments

`packageName_s_1`
: a `string` that is the package name to seek for

`libraryPath_s_1`
: a `string` that is the folder to scrutinize

Value

A `vector` of function names

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
# take to much time on Windows apparently to pass CRAN package acceptance tests
if (.Platform$OS.type == "unix")
  retrievePackageFunctionNames('wyz.code.offensiveProgramming')
```

`retrieveTestCaseDefinitions` *Retrieve test case definitions or test case descriptions.*

Description

From an instrumented class, retrieve the test case definitions or descriptions.

Usage

```
retrieveTestCaseDefinitions(object_o_1)
retrieveTestCaseDescriptions(object_o_1)
```

Arguments

`object_o_1`	the `object` to consider

Value

For `function`, `retrieveTestCaseDefinitions`, 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

For `function`, `retrieveTestCaseDescriptions`, either a character `vector` when no test case definitions exists or a `data.table` of the test case descriptions.

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

retrieveTestCaseDescriptions(Addition())
retrieveTestCaseDescriptions(AdditionTCPartial())
```

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)
```

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

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)
```

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 `string` that is a semantic parameter name, to express expected function return type

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 entry character vector, that is the test case description string

Value

An object that is an R environment class.

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

Verifies class name compliance with a policy.

Usage

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

Arguments

name_s	a string that is 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

TRUE when name complies with policy, FALSE otherwise.

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)
```

verifyFunctionArguments *Verify Function Arguments*

Description

Use this function to verify function arguments.

Usage

```
verifyFunctionArguments(arguments_l, abort_b_1 = TRUE, verbosity_b_1 = FALSE)
```

Arguments

arguments_l	An unconstrained list, representing the arguments. Should always result from a call to mget(ls()).
abort_b_1	A single boolean value stating if processing abortion should be triggered in case of error
verbosity_b_1	A single boolean value.

Details

This function allows to check all parameter types and values in a single line of code.

See examples below to know how to put this function in action.

Value

Returned value depends on parameter abort_b_1 value.

When set to TRUE, any error will abort processing by issuing a call to stop function.

When set to FALSE, returned value is a boolean. It is TRUE only when no error have been detected. Otherwise FALSE.

Note

This function whenever used, should be the first statement of your function code.

Using this function outside function code is a non-sense.

Author(s)

Fabien Gelineau <neonira@gmail.com>

Maintainer: Fabien Gelineau <neonira@gmail.com>

Examples

```
fun <- function(values_i_3m) {
  verifyFunctionArguments(mget(ls()), FALSE, FALSE)
}

fun(1)
# [1] FALSE

fun(1:7)
# [1] TRUE

nonOPFun <- function(x) {
  verifyFunctionArguments(mget(ls()), FALSE, TRUE)
}

nonOPFun(1:7)
# $x
# [1] 1 2 3 4 5 6 7
#
# x FALSE unknown suffix, [NA]
#
# [1] FALSE

# real use case with abortion
myFunWithAbortion <- function(values_i_3m) {
  verifyFunctionArguments(mget(ls()))
  # ...
}

tryCatch(myFunWithAbortion(1), error = function(e) cat(e$message, '\n'))
# argument mistmatch [values_i_3m] wrong length, was expecting [3m] , got [1]

# real use case without abortion
myFunWithoutAbortion <- function(values_i_3m) {
  if (!verifyFunctionArguments(mget(ls()), FALSE)) return(FALSE)
  cat('continuing processing ...\n')
  TRUE
}

myFunWithoutAbortion(1)
# FALSE

myFunWithoutAbortion(1:3)
# continuing processing ...
# TRUE
```

---

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

Verifies your declared return type definitions and detects anomalies.

Usage

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

Arguments

`object_o_1`	The `object` to be considered
`requiresFullInstrumentation_b_1`	a `boolean` stating if full instrumentation is required

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 executed 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)
```

Arguments

`object_o_1`	the object to be checked

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 verifyClassName and 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

Checks for test cases definition compliance and dectects uncompliances.

Usage

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

Arguments

object_o_1
: The objectto be considered

requiresFullInstrumentation_b_1
: a boolean stating if full instrumentation is required

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