Package: ospsuite.utils (via r-universe)

July 6, 2026

Type Package

Title Utility Functions for Open Systems Pharmacology R Packages

Version 1.11.1

Maintainer Michael Sevestre <michael@design2code.ca>

Description A collection of utility functions for R packages in the Open Systems Pharmacology ecosystem. Contains helper functions for working with R6 objects, enumerated lists, and text formatting. Additionally, it provides functions to validate argument inputs.

License GPL-2

URL https://github.com/open-systems-pharmacology/OSPSuite.RUtils, https://www.open-systems-pharmacology.org/OSPSuite.RUtils/

BugReports https://github.com/open-systems-pharmacology/OSPSuite.RUtils/issues

Depends R (>= 4.4)

Imports purrr, R6, stringi, cli, lifecycle, logger, crayon, glue

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Collate 'enum.R' 'formatNumerics.R' 'logger.R' 'messages.R' 'objectCount.R' 'osp_print.R' 'utilities.R' 'ospsuite.utils-env.R' 'printable.R' 'utilities-conditional.R' 'utilities-logger.R' 'utilities-validation.R' 'validation-emptiness.R' 'validation-enum.R' 'validation-expected-length.R' 'validation-file-extensions.R' 'validation-inclusion.R' 'validation-options.R' 'validation-paths.R' 'validation-same-length.R' 'validation-type.R' 'validation-uniqueness.R' 'validation-utf.R' 'validation-vector.R'

VignetteBuilder knitr

Config/pak/sysreqs libicu-dev

Repository https://open-systems-pharmacology.r-universe.dev

Date/Publication 2026-06-01 10:13:06 UTC

RemoteUrl https://github.com/Open-Systems-Pharmacology/OSPSuite.RUtils

RemoteRef v1.11.1

RemoteSha 6d22617df457ec6e89b21bd25a99bb3080e4fb6c

Contents

`characterOption` *Create Character Option Specification*

Description

Create Character Option Specification

Usage

```
characterOption(
  allowedValues = NULL,
  nullAllowed = FALSE,
  naAllowed = FALSE,
  expectedLength = 1
)
```

Arguments

`allowedValues`	Vector of permitted values. Defaults to `NULL` (any value allowed).
`nullAllowed`	Logical flag indicating whether `NULL` is permitted. Defaults to `FALSE`.
`naAllowed`	Logical flag indicating whether `NA` values are permitted. Defaults to `FALSE`.
`expectedLength`	Expected length of the option value. Use `NULL` for any length, `1` for scalar (default), or a positive integer for specific length.

Value

An S3 object of class `optionSpec_character` and `optionSpec`.

`cliFormat` *cliFormat*

Description

Format text into cli inline format Allows evaluation of expressions within the text before submitting to logs

Usage

```
cliFormat(..., .envir = parent.frame())
```

Arguments

`...`	Characters to format
`.envir`	Environment in which to evaluate the expressions

Value

A formatted character string

`enum` *Define an enumerated list*

Description

Create an enumeration to be used instead of arbitrary values in code. In some languages (C, C++, Python, etc.), enum (or enumeration) is a data type that consists of integer constants and is ideal in contexts where a variable can take on only one of a limited set of possible values (e.g. day of the week). Since R programming language natively doesn't support enumeration, the current function provides a way to create them using lists.

Usage

```
enum(enumValues)
```

Arguments

`enumValues`	A vector or a list of comma-separated constants to use for creating the enum. Optionally, these can be named constants.

Value

An enumerated list.

See Also

Other enumeration-helpers: `enumGetKey()`, `enumGetValue()`, `enumHasKey()`, `enumKeys()`, `enumPut()`, `enumRemove()`, `enumValues()`

Examples

```
# Without predefined values
Color <- enum(c("Red", "Blue", "Green"))
Color
myColor <- Color$Red
myColor

# With predefined values
Symbol <- enum(c(Diamond = 1, Triangle = 2, Circle = 2))
Symbol

mySymbol <- Symbol$Diamond
mySymbol
```

---

enumGetKey *Get the key mapped to the given value in an* enum

---

Description

Get the key mapped to the given value in an enum

Usage

```
enumGetKey(enum, value)

getEnumKey(enum, value)
```

Arguments

enum	The enum where the key-value pair is stored
value	The value that is mapped to the key

Value

Key under which the value is stored. If the value is not in the enum, NULL is returned.

See Also

Other enumeration-helpers: `enum()`, `enumGetValue()`, `enumHasKey()`, `enumKeys()`, `enumPut()`, `enumRemove()`, `enumValues()`

Examples

```
Symbol <- enum(c(Diamond = 1, Triangle = 2, Circle = 2))
enumGetKey(Symbol, 1)
```

`enumGetValue` *Get enum values*

Description

Return the value that is stored under the given key. If the key is not present, an error is thrown.

Usage

```
enumGetValue(enum, key)
```

Arguments

`enum`	The `enum` that contains the key-value pair.
`key`	The `key` under which the value is stored.

Value

Value that is assigned to `key`.

See Also

Other enumeration-helpers: `enum()`, `enumGetKey()`, `enumHasKey()`, `enumKeys()`, `enumPut()`, `enumRemove()`, `enumValues()`

Examples

```
Symbol <- enum(c(Diamond = 1, Triangle = 2, Circle = 2))
enumGetValue(Symbol, "Diamond")
```

`enumHasKey` *Check if an enum has a certain key.*

Description

Check if an enum has a certain key.

Usage

```
enumHasKey(key, enum)
```

Arguments

key	Key to check for.
enum	Enum where to look for the key.

Value

TRUE if a key-value pair for key exists, FALSE otherwise.

See Also

Other enumeration-helpers: enum(), enumGetKey(), enumGetValue(), enumKeys(), enumPut(), enumRemove(), enumValues()

Examples

```
Symbol <- enum(c(Diamond = 1, Triangle = 2, Circle = 2))
enumHasKey("Diamond", Symbol)
enumHasKey("Square", Symbol)
```

---

enumKeys *Get all keys of an enum*

Description

Get all keys of an enum

Usage

```
enumKeys(enum)
```

Arguments

enum	enum containing the keys.

Value

List of key names.

See Also

Other enumeration-helpers: enum(), enumGetKey(), enumGetValue(), enumHasKey(), enumPut(), enumRemove(), enumValues()

Examples

```
Symbol <- enum(c(Diamond = 1, Triangle = 2, Circle = 2))
enumKeys(Symbol)
```

enumPut *Add a new key-value pairs to an* enum

Description

Add a new key-value pairs to an enum

Usage

```
enumPut(keys, values, enum, overwrite = FALSE)
```

Arguments

keys	Keys of the values to be added
values	Values to be added
enum	The enum to which the specified key-value pairs should be added. **WARNING**: the original object is **not** modified!
overwrite	If TRUE and a value with any of the given keys exists, it will be overwritten with the new value. Otherwise, an error is thrown. Default is FALSE.

Value

Enum with added key-value pair.

See Also

Other enumeration-helpers: enum(), enumGetKey(), enumGetValue(), enumHasKey(), enumKeys(), enumRemove(), enumValues()

Examples

```
myEnum <- enum(c(a = "b"))
myEnum <- enumPut("c", "d", myEnum)
myEnum <- enumPut(c("c", "d", "g"), c(12, 2, "a"), myEnum, overwrite = TRUE)
```

enumRemove *Remove an entry from the enum.*

Description

Remove an entry from the enum.

Usage

```
enumRemove(keys, enum)
```

Arguments

keys	Key(s) of entries to be removed from the enum
enum	Enum from which the entries to be removed **WARNING**: the original object is not modified!

Value

Enum without the removed entries.

See Also

Other enumeration-helpers: enum(), enumGetKey(), enumGetValue(), enumHasKey(), enumKeys(), enumPut(), enumValues()

Examples

```
Symbol <- enum(c(Diamond = 1, Triangle = 2, Circle = 2))

# either by key
enumRemove("Diamond", Symbol)

# or by position
enumRemove(2L, Symbol)
```

---

enumValues *Get the values stored in an enum*

Description

Get the values stored in an enum

Usage

```
enumValues(enum)
```

Arguments

enum	enum containing the values

Value

List of values stored in the enum.

See Also

Other enumeration-helpers: enum(), enumGetKey(), enumGetValue(), enumHasKey(), enumKeys(), enumPut(), enumRemove()

Examples

```
Symbol <- enum(c(Diamond = 1, Triangle = 2, Circle = 2))
enumValues(Symbol)
```

`flattenList` *Flatten a list to an atomic vector of desired type*

Description

Flatten a list to an atomic vector of desired type

Usage

```
flattenList(x, type)
```

Arguments

`x`	A list or an atomic vector. If the latter, no change will be made.
`type`	Type of atomic vector to be returned.

Details

The `type` argument will decide which variant from `purrr::flatten()` family is used to flatten the list.

Value

An atomic vector of desired type.

Examples

```
flattenList(list(1, 2, 3, NA), type = "numeric")
flattenList(list(TRUE, FALSE, NA), type = "integer")
```

foldSafe *Safe fold calculation*

Description

Calculates x / y while substituting values below epsilon (for x and y) by epsilon. x and y must be of the same length

Usage

```
foldSafe(x, y, epsilon = ospsuiteUtilsEnv$LOG_SAFE_EPSILON)
```

Arguments

x	A numeric or a vector of numerics.
y	A numeric or a vector of numerics.
epsilon	A very small number which is considered as threshold below which all values are treated as epsilon. Allows computation of fold changes for values close to 0. Default value is getOSPSuiteUtilsSetting("LOG_SAFE_EPSILON").

Value

A vector with x / y.

Examples

```
inputX <- c(NA, 1, 5, 0, -1)
inputY <- c(1, -1, NA, 0, -1)
folds <- foldSafe(inputX, inputY)
```

formatNumerics *formatNumerics*

Description

Render numeric values of an object as character using the specified format:

- If object is a data.frame or a list, formatNumerics applies on each of its fields
- If object is of type character or integer, formatNumerics renders the values as is
- If object is of type numeric, formatNumerics applies the defined format

Usage

```
formatNumerics(
  object,
  digits = ospsuiteUtilsEnv$formatNumericsDigits,
  scientific = FALSE
)
```

Arguments

object	An R object such as a list, a data.frame, character or numeric values.
digits	Number of decimal digits to render
scientific	Logical value defining if scientific writing is rendered

Value

Numeric values are rendered as character values. If object is a data.frame or a list, a data.frame or list is returned with numeric values rendered as character values.

Examples

```
# Format array of numeric values
formatNumerics(log(c(12, 15, 0.3)), digits = 1, scientific = TRUE)

# Format a data.frame
x <- data.frame(parameter = c("a", "b", "c"), value = c(1, 110.4, 6.666))
formatNumerics(x, digits = 2, scientific = FALSE)
```

---

getLogFolder *getLogFolder*

Description

Get current log folder where logs are saved

Usage

```
getLogFolder()
```

Examples

```
## Not run:
# Set/get log folder to a temporary directory
setLogFolder(tempdir())
getLogFolder()

# Set/get logFolder to `NULL`, cancel saving of logs
setLogFolder()
getLogFolder()
```

```
## End(Not run)
```

`getOSPSuiteUtilsSetting` *Get the value of a global* `{ospsuite.utils}` *package setting.*

Description

Get the value of a global `{ospsuite.utils}` package setting.

Usage

```
getOSPSuiteUtilsSetting(settingName)
```

Arguments

`settingName`	String name of the setting

Value

Value of the setting stored in `ospsuiteEnv`. If the setting does not exist, an error is thrown.

Examples

```
getOSPSuiteUtilsSetting("packageName")
getOSPSuiteUtilsSetting("suiteName")
getOSPSuiteUtilsSetting("formatNumericsDigits")
```

`hasEmptyStrings` *Validate that no empty string is present*

Description

Validate that no empty string is present

Usage

```
hasEmptyStrings(x)

validateHasOnlyNonEmptyStrings(x)
```

Arguments

`x`	A character string or a vector of character strings.

Details

If any of the following conditions are met, the input string is considered empty:

- if any NAs are present (e.g. `x = c("a", "abc", NA)`)
- if string is empty (e.g. `x = list("a", "abc", "")`)
- if length is 0 (e.g. `x = character()`)

Value

- `hasEmptyStrings()` returns `TRUE` if any of the strings are empty; `FALSE` otherwise.
- `validateHasOnlyNonEmptyStrings()` produces an error if empty string are present. It returns `NULL` otherwise.

Examples

```
hasEmptyStrings(c("x", "y")) # FALSE
hasEmptyStrings(list("x", "y")) # FALSE
hasEmptyStrings("   abc   ") # FALSE
hasEmptyStrings(c("", "y")) # TRUE
hasEmptyStrings(list("", "y")) # TRUE
hasEmptyStrings(NA) # TRUE
hasEmptyStrings(character(0)) # TRUE
hasEmptyStrings(c(NA, "x", "y")) # TRUE

validateHasOnlyNonEmptyStrings(c("x", "y")) # NULL
validateHasOnlyNonEmptyStrings(list("x", "y")) # NULL
validateHasOnlyNonEmptyStrings("   abc   ") # NULL
# validateHasOnlyNonEmptyStrings(c("", "y")) # error
# validateHasOnlyNonEmptyStrings(list("", "y")) # error
# validateHasOnlyNonEmptyStrings(NA) # error
# validateHasOnlyNonEmptyStrings(character(0)) # error
# validateHasOnlyNonEmptyStrings(c(NA, "x", "y")) # error
```

---

hasOnlyDistinctValues *Validate that a vector has only unique values*

Description

Validate that a vector has only unique values

Usage

```
hasOnlyDistinctValues(values, na.rm = TRUE)

validateHasOnlyDistinctValues(values, na.rm = TRUE)
```

Arguments

values	An array of values
na.rm	Logical to decide if missing values should be removed from the duplicate checking. Note that duplicate NA values are flagged if na.rm=FALSE.

Value

- hasOnlyDistinctValues returns TRUE if all values are unique.
- validateHasOnlyDistinctValues() returns NULL if only unique values present, otherwise produces error.

Examples

```
hasOnlyDistinctValues(c("x", "y"))
hasOnlyDistinctValues(c("x", "y", "x"))
hasOnlyDistinctValues(c("x", NA, "y", NA), na.rm = FALSE)
hasOnlyDistinctValues(c("x", NA, "y", NA), na.rm = TRUE)

validateHasOnlyDistinctValues(c("x", "y")) # NULL
# validateHasOnlyDistinctValues(c("x", "y", "x")) # error
```

---

ifEqual *Value conditional on equality*

Description

Short-key checking if arguments 1 and 2 are equal, output argument 3 if equal, or output argument 4 otherwise.

Usage

```
ifEqual(x, y, outputIfEqual, outputIfNotEqual = NULL)
```

Arguments

x	argument 1
y	argument 2
outputIfEqual	argument 3
outputIfNotEqual	argument 4

Examples

```
ifEqual(1, 1, "x", "y") # "x"
ifEqual(1, 2, "x", "y") # "y"
```

ifIncluded *Value conditional on inclusion*

Description

Shortkey checking if arguments 1 is included in 2, output argument 3 if included, or output argument 4 otherwise.

Usage

```
ifIncluded(x, y, outputIfIncluded, outputIfNotIncluded = NULL)
```

Arguments

x	argument 1
y	argument 2
outputIfIncluded	argument 3
outputIfNotIncluded	argument 4

Examples

```
ifIncluded("a", c("a", "b"), 1, 2) # 1
ifIncluded("x", c("a", "b"), 1, 2) # 2
```

ifNotNull *Value conditional on* NULL

Description

Short-key checking if argument 1 is not NULL, output the argument 2 if not null, or output argument 3 otherwise.

Check if condition is not NULL, if so output outputIfNotNull, otherwise, output outputIfNull.

Usage

```
ifNotNull(condition, outputIfNotNull, outputIfNull = NULL)
```

Arguments

condition	argument 1
outputIfNotNull	argument 2
outputIfNull	argument 3

Value

outputIfNotNull if condition is not NULL, outputIfNull otherwise.

Examples

```
ifNotNull(NULL, "x")
ifNotNull(NULL, "x", "y")
ifNotNull(1 < 2, "x", "y")
```

---

integerOption *Create Integer Option Specification*

Description

Create Integer Option Specification

Usage

```
integerOption(
  min = -Inf,
  max = Inf,
  nullAllowed = FALSE,
  naAllowed = FALSE,
  expectedLength = 1
)
```

Arguments

min	Minimum allowed value. Defaults to -Inf.
max	Maximum allowed value. Defaults to Inf.
nullAllowed	Logical flag indicating whether NULL is permitted. Defaults to FALSE.
naAllowed	Logical flag indicating whether NA values are permitted. Defaults to FALSE.
expectedLength	Expected length of the option value. Use NULL for any length, 1 for scalar (default), or a positive integer for specific length.

Value

An S3 object of class optionSpec_integer and optionSpec.

isEmpty *Validate if the provided object is empty*

Description

Validate if the provided object is empty

Usage

```
isEmpty(object)

validateIsNotEmpty(object)
```

Arguments

object	An object or an atomic vector or a list of objects.

Value

- isEmpty() returns TRUE if the object is empty; FALSE otherwise.
- validateIsNotEmpty() returns NULL if validation is successful. Otherwise, error is signaled.

Examples

```
# empty list or data.frame
isEmpty(NULL)
isEmpty(numeric())
isEmpty(list())
isEmpty(data.frame())

# accounts for filtering of arrays and data.frame
df <- data.frame(x = c(1, 2, 3), y = c(4, 5, 6))
isEmpty(df)
isEmpty(df$x[FALSE])
isEmpty(df[FALSE, ])

# validation helper
validateIsNotEmpty(list(1, 2)) # NULL
# validateIsNotEmpty(NULL) # error
```

isFileExtension *Validate if the provided path has required extension*

Description

Validate if the provided path has required extension

Usage

```
isFileExtension(file, extension)

validateIsFileExtension(file, extension)
```

Arguments

file	A name of the file or full path.
extension	A required extension of the file.

Value

`isFileExtension()` returns `TRUE` if the file name (or full path) includes the extension.

If validations are successful, `validateIsFileExtension()` returns `NULL`. Otherwise, error is signaled.

Examples

```
isFileExtension("enum.R", "R") # TRUE
isFileExtension("enum.R", "pkml") # FALSE

validateIsFileExtension("enum.R", "R") # NULL
# validateIsFileExtension("enum.R", "pkml") # error
```

isFileUTF8 *Assess if a file is UTF-8 encoded.*

Description

Assess if a file is UTF-8 encoded.

Usage

```
isFileUTF8(file)
```

Arguments

file	A name of the file or full path.

Value

A logical assessing whether there is non UTF-8 encoded characters in `file`

Examples

```
writeLines(c("Hello, world!"), "utf.txt")
writeLines(c("Hello, world!", "\xb5g/L"), "non-utf.txt")

isFileUTF8("utf.txt") # TRUE
isFileUTF8("non-utf.txt") # FALSE
```

---

isIncluded *Check if a vector of values is included in another vector of values*

Description

Check if a vector of values is included in another vector of values

Usage

```
isIncluded(values, parentValues)

validateIsIncluded(values, parentValues, nullAllowed = FALSE)
```

Arguments

`values`	A vector of values.
`parentValues`	A vector of values where `values` are checked for inclusion.
`nullAllowed`	Boolean flag if `NULL` is accepted for the `object`. If `TRUE`, `NULL` always returns `TRUE`, otherwise `NULL` returns `FALSE`. Default is `FALSE`.

Value

- `isIncluded()` returns `TRUE` if the value or **all** `values` (if it's a vector) are present in the `parentValues`; `FALSE` otherwise.
- `validateIsIncluded()` returns `NULL` if child value is included in parent value set, otherwise error is signaled.

Examples

```
# check if a column is present in dataframe
A <- data.frame(
  col1 = c(1, 2, 3),
  col2 = c(4, 5, 6),
  col3 = c(7, 8, 9)
)
isIncluded("col3", names(A)) # TRUE

# check if single element is present in a vector (atomic or non-atomic)
isIncluded("x", list("w", "x", 1, 2)) # TRUE
isIncluded("x", c("w", "a", "y")) # FALSE

# check if **all** values (if it's a vector) are contained in parent values
isIncluded(c("x", "y"), c("a", "y", "b", "x")) # TRUE
isIncluded(list("x", 1), list("a", "b", "x", 1)) # TRUE
isIncluded(c("x", "y"), c("a", "b", "x")) # FALSE
isIncluded(list("x", 1), list("a", "b", "x")) # FALSE

# corresponding validation
validateIsIncluded("col3", names(A)) # NULL
# validateIsIncluded("col6", names(A)) # error
```

`isOfLength` *Check if the provided object has expected length*

Description

Check if the provided object has expected length

Usage

```
isOfLength(object, nbElements)

validateIsOfLength(object, nbElements)
```

Arguments

`object`	An object or a list of objects
`nbElements`	number of elements that are supposed in object

Value

- `isOfLength()` returns `TRUE` if the object or all objects inside the list have `nbElements`.
- For `validateIsOfLength()`, if validations are successful, `NULL` is returned. Otherwise, error is signaled.

Note

Only the first level of the given list is considered.

Examples

```
df <- data.frame(x = c(1, 2, 3))

isOfLength(df, 1) # TRUE
isOfLength(df, 3) # FALSE

validateIsOfLength(list(1, 2), 2L) # NULL
# validateIsOfLength(c("3", "4"), 3L) # error
```

`isOfType` *Check if the provided object is of certain type*

Description

Check if the provided object is of certain type

Usage

```
isOfType(object, type, nullAllowed = FALSE)
```

Arguments

`object`	An object or an atomic vector or a list of objects.
`type`	A single string or a vector of string representation or class of the type that should be checked for.
`nullAllowed`	Boolean flag if `NULL` is accepted for the `object`. If `TRUE`, `NULL` always returns `TRUE`, otherwise `NULL` returns `FALSE`. Default is `FALSE`.

Value

`TRUE` if the object or all objects inside the list are of the given type.

Note

Only the first level of the given list is considered.

Examples

```
# checking type of a single object
df <- data.frame(x = c(1, 2, 3))
isOfType(df, "data.frame")
```

isPathAbsolute *Check if path is absolute*

Description

Relative paths will be detected based on the presence of wildcard character(*) in the path specification.

Usage

```
isPathAbsolute(path)

validateIsPathAbsolute(path)

validatePathIsAbsolute(path)
```

Arguments

path	A valid file path name.

Value

- `isPathAbsolute()` returns `TRUE` if path is absolute (no wildcard); `FALSE` otherwise.
- `validateIsPathAbsolute()` returns `NULL` if path is absolute. Otherwise, error is signaled.

Examples

```
# check if path is absolute
isPathAbsolute("Organism|path") # TRUE
isPathAbsolute("Organism|*path") # FALSE

# validation: no error if path is absolute
validateIsPathAbsolute("Organism|path")

# validation: error otherwise
# validateIsPathAbsolute("Organism|*path")
```

isSameLength *Validate if objects are of same length*

Description

Validate if objects are of same length

Usage

```
isSameLength(...)

validateIsSameLength(...)
```

Arguments

...	Objects to compare.

Value

- `isSameLength()` returns `TRUE` if all objects have same lengths.
- For `validateIsSameLength()`, if validations are successful, `NULL` is returned. Otherwise, error is signaled.

Examples

```
# compare length of only 2 objects
isSameLength(mtcars, ToothGrowth) # FALSE
isSameLength(cars, BOD) # TRUE

# or more number of objects
isSameLength(c(1, 2), c(TRUE, FALSE), c("x", "y")) # TRUE
isSameLength(list(1, 2), list(TRUE, FALSE), list("x")) # FALSE

# validation
validateIsSameLength(list(1, 2), c("3", "4")) # NULL
# validateIsSameLength(list(1, 2), c("3", "4"), c(FALSE)) # error
```

---

isUTF8 *Assess if a character vector is UTF-8 encoded.*

Description

Assess if a character vector is UTF-8 encoded.

Usage

```
isUTF8(text)
```

Arguments

text	A character vector

Value

A logical assessing whether there is non UTF-8 encoded characters in `text`

Examples

```
isUTF8("Hello, world!") # TRUE
isUTF8("\xb5g/L") # FALSE
```

`logCatch` *logCatch*

Description

Catch errors, log and display meaningful information

Usage

```
logCatch(expr)
```

Arguments

`expr`	Evaluated code chunks

Examples

```
# Catch and display warning message
logCatch({
  warning("This is a warning message")
})
```

`logDebug` *logDebug*

Description

Log debug with time stamp

Usage

```
logDebug(msg)
```

Arguments

`msg`	Character values of message to log that leverages `cli` formatting.

Examples

```
# Log debug
logDebug("This is a debugging message")
```

logError *logError*

Description

Log error with time stamp

Usage

```
logError(msg)
```

Arguments

msg	Character values of message to log that leverages cli formatting.

Examples

```
# Log error
logError(cliFormat("This is an {.strong error} message"))

# Log error with indications
logError(cliFormat(
  "This is an {.strong error} message",
  "Check these {.val values} or this {.fn function}"
))
```

logicalOption *Create Logical Option Specification*

Description

Create Logical Option Specification

Usage

```
logicalOption(nullAllowed = FALSE, naAllowed = FALSE, expectedLength = 1)
```

Arguments

nullAllowed	Logical flag indicating whether NULL is permitted. Defaults to FALSE.
naAllowed	Logical flag indicating whether NA values are permitted. Defaults to FALSE.
expectedLength	Expected length of the option value. Use NULL for any length, 1 for scalar (default), or a positive integer for specific length.

Value

An S3 object of class optionSpec_logical and optionSpec.

logInfo *logInfo*

Description

Log information with time stamp accounting for message type. Message type will point toward the most appropriate `cli` function display.

Usage

```
logInfo(msg, type = "info")
```

Arguments

`msg`	Character values of message to log that leverages `cli` formatting.
`type`	Name of the message type to toward best `cli` display: • `"info"`: uses `cli::cli_alert_info()` • `"success"`: uses `cli::cli_alert_success()` • `"h1"`: uses `cli::cli_h1()` • `"h2"`: uses `cli::cli_h2()` • `"h3"`: uses `cli::cli_h3()` • `"text"`: uses `cli::cli_text()` • `"alert"`: uses `cli::cli_alert()` • `"li"`: uses `cli::cli_li()` • `"ol"`: uses `cli::cli_ol()` • `"progress_step"`: uses `cli::cli_progress_step()`

Examples

```
# Log information
logInfo(cliFormat("This is an {.strong info} message"))

# Log a title
logInfo(cliFormat("Task: {.strong tic toc test}"), type = "h1")

# Log success
t0 <- tic()
Sys.sleep(3)
logInfo(cliFormat("Task: {.strong tic toc test} completed [{toc(t0, \"s\")}]"), type = "success")
```

logSafe *Computes logarithm of a number or of a vector of numbers and handles zeros while substituting all values below* epsilon *by* epsilon.

Description

Computes logarithm of a number or of a vector of numbers and handles zeros while substituting all values below epsilon by epsilon.

Usage

```
logSafe(x, base = exp(1), epsilon = ospsuiteUtilsEnv$LOG_SAFE_EPSILON)
```

Arguments

x	A numeric or a vector of numerics.
base	a positive or complex number: the base with respect to which logarithms are computed. Defaults to e = exp(1).
epsilon	A very small number which is considered as threshold below which all values are treated as epsilon. Allows computation of log close to 0. Default value is getOSPSuiteUtilsSetting("LOG_SAFE_EPSILON").

Value

log(x, base = base) for x > epsilon, or log(epsilon, base = base), or NA_real_ for NA elements.

Examples

```
inputVector <- c(NA, 1, 5, 0, -1)
logSafe(inputVector)
```

logWarning *logWarning*

Description

Log warning with time stamp

Usage

```
logWarning(msg)
```

Arguments

msg	Character values of message to log that leverages cli formatting.

Examples

```
# Log warning
logWarning(cliFormat("This is a {.strong warning} message"))
```

`messages` *List of functions and strings used to signal error messages*

Description

Most of these messages will be relevant only in the context of OSP R package ecosystem.

Usage

```
messages
```

Format

An object of class `list` of length 47.

Value

A string with error message.

Examples

```
# example with string
messages$errorEnumNotAllNames

# example with function
messages$errorPropertyReadOnly("age")

# example display with warning
warning(messages$errorPropertyReadOnly("age"))

# example display using logs
logInfo(messages$errorPropertyReadOnly("age"))
```

numericOption *Create Numeric Option Specification*

Description

Create Numeric Option Specification

Usage

```
numericOption(
  min = -Inf,
  max = Inf,
  nullAllowed = FALSE,
  naAllowed = FALSE,
  expectedLength = 1
)
```

Arguments

min	Minimum allowed value. Defaults to -Inf.
max	Maximum allowed value. Defaults to Inf.
nullAllowed	Logical flag indicating whether NULL is permitted. Defaults to FALSE.
naAllowed	Logical flag indicating whether NA values are permitted. Defaults to FALSE.
expectedLength	Expected length of the option value. Use NULL for any length, 1 for scalar (default), or a positive integer for specific length.

Value

An S3 object of class optionSpec_numeric and optionSpec.

objectCount *Count number of objects*

Description

Count number of objects

Usage

```
objectCount(x)
```

Arguments

x	An object (an atomic vector, a list, or instance(s) of a class).

Details

Classes in R can roughly be distinguished as following:

- Vector-style classes have the property that `length()` represents number of elements (e.g., `factor`, `list`, etc.).
- Record-style (or dataframe or scalar) classes, on the other hand, have complex structures to represent a single thing (e.g., `data.frame`, `R6`, `lm`, etc.).

This function counts objects differently depending on the entered class:

If the argument is a vector or a vector-style class, it will return the output from `length()` function. Otherwise, it will returns `1`.

For example,

- `length(mtcars)` returns `11`, but `objectCount(mtcars)` will return `1`, while
- `length(list(1, 2))` returns `2`, and `objectCount(list(1, 2))` will return `2` as well.

Value

Integer representing the count of objects.

Examples

```
# vectors or vector-style classes
objectCount(c(1, 2, 3)) # 3
objectCount(list("a", "b")) # 2
objectCount(list(iris, mtcars)) # 2

# everything else
objectCount(mtcars) # 1
objectCount(lm(wt ~ mpg, mtcars)) # 1
objectCount(new.env()) # 1
```

`ospPrintClass` *Print an object's class name*

Description

Prints the class name of an object with nice formatting using cli.

Usage

```
ospPrintClass(x)
```

Arguments

`x`	An R object

Value

Invisibly returns the input object

See Also

Other print functions: `ospPrintHeader()`, `ospPrintItems()`

Examples

```
# Print class name of a data frame
ospPrintClass(iris)
```

`ospPrintHeader` *Print a header with specified level*

Description

Prints a header with the specified level (H1, H2, or H3) using cli.

Usage

```
ospPrintHeader(text, level = 1)
```

Arguments

`text`	The text to print as a header
`level`	The header level (1, 2, or 3)

Value

Invisibly returns NULL

See Also

Other print functions: `ospPrintClass()`, `ospPrintItems()`

Examples

```
# Print different header levels
ospPrintHeader("Main Title", 1)
ospPrintHeader("Section Title", 2)
ospPrintHeader("Subsection Title", 3)
```

`ospPrintItems` *Print a list of items with an optional title*

Description

Prints a list of items from a vector or list, with an optional title. Items are indented and prefixed with a dash.

Usage

```
ospPrintItems(x, title = NULL, print_empty = FALSE)
```

Arguments

`x`	A vector or list
`title`	Optional title to display before the list (default: NULL)
`print_empty`	Whether to print empty values (NULL, NA, empty string) (default: FALSE)

Value

Invisibly returns the input object

See Also

Other print functions: `ospPrintClass()`, `ospPrintHeader()`

Examples

```
# Print a simple vector with title
vector <- c("A", "B", "C")
ospPrintItems(vector, title = "Letters")

# Print a named vector with title
named_vector <- c(A = 1, B = 2, C = 3)
ospPrintItems(named_vector, title = "Letters")

# Print a list including empty values
list_with_nulls <- list("Min" = NULL, "Max" = 100, "Unit" = NA)
ospPrintItems(list_with_nulls, title = "Parameters", print_empty = TRUE)
```

ospsuiteUtilsSettingNames *Names of the settings stored in* ospsuiteEnv. *Can be used with* getOSPSuiteUtilsSetting()

Description

Names of the settings stored in ospsuiteEnv. Can be used with getOSPSuiteUtilsSetting()

Usage

```
ospsuiteUtilsSettingNames
```

Format

An object of class list of length 5.

Examples

```
ospsuiteUtilsSettingNames
```

Printable *Printable*

Description

Base class that implements some basic properties for printing to console.

Methods

Public methods:

- Printable$new()
- Printable$clone()

Method new(): Create a new Printable object.

Usage:

```
Printable$new()
```

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
Printable$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
myPrintable <- R6::R6Class(
  "myPrintable",
  inherit = Printable,
  public = list(
    x = NULL,
    y = NULL,
    print = function() {
      private$printClass()
      private$printLine("x", self$x)
      private$printLine("y", self$y)
      invisible(self)
    }
  )
)

x <- myPrintable$new()
x
```

`setErrorMasking` *setErrorMasking*

Description

Mask error trace messages

Usage

```
setErrorMasking(patterns)
```

Arguments

`patterns`	Character patterns to identify with `grepl()` when masking messages

Examples

```
## Not run:
setErrorMasking(c("tryCatch", "withCallingHandlers"))

## End(Not run)
```

setInfoMasking *setInfoMasking*

Description

Mask info messages

Usage

```
setInfoMasking(patterns)
```

Arguments

patterns	Character patterns to identify with `grepl()` when masking messages

Examples

```
## Not run:
# Mask ggplot2 message when line is used with 1 value per group
setInfoMasking("Each group consists of only one observation")

## End(Not run)
```

setLogFolder *setLogFolder*

Description

Initialize logs and their settings

Usage

```
setLogFolder(logFolder = NULL)
```

Arguments

logFolder	Optional folder path to save log file

setWarningMasking *setWarningMasking*

Description

Mask warning messages

Usage

```
setWarningMasking(patterns)
```

Arguments

patterns	Character patterns to identify with grepl() when masking messages

Examples

```
## Not run:
# Mask ggplot2 warning message when missing values are found
setWarningMasking("rows containing missing values")

## End(Not run)
```

tic *tic*

Description

Trigger time tracker

Usage

```
tic()
```

Value

System time

Examples

```
tic()
```

timeStamp *timeStamp*

Description

Print time stamp

Usage

```
timeStamp()
```

Examples

```
timeStamp()
```

toc *toc*

Description

Get elapsed time between `tic` trigger and now

Usage

```
toc(tic, unit = "min")
```

Arguments

`tic`	Start time
`unit`	display unit of elapsed time

Value

Character displaying elapsed time in `unit`

Examples

```
t0 <- tic()
Sys.sleep(2)
# Get elapsed time in seconds
toc(t0, "s")
# Get elapsed time in minutes
toc(t0, "min")
```

toList *Make sure the object is a list*

Description

Make sure the object is a list

Usage

```
toList(object)
```

Arguments

object	Object to be converted to a list.

Value

If `is.list(object) == TRUE`, returns the `object`; otherwise, `list(object)`.

Examples

```
toList(list("a" = 1, "b" = 2))
toList(c("a" = 1, "b" = 2))
```

toMissingOfType *Convert special constants to* NA *of desired type*

Description

Convert special constants to NA of desired type

Usage

```
toMissingOfType(x, type)
```

Arguments

x	A single element.
type	Type of atomic vector to be returned.

Details

Special constants (`NULL`, `Inf`, `-Inf`, `NaN`, `NA`) will be converted to `NA` of desired type.

This function is **not** vectorized, and therefore only scalar values should be entered.

Examples

```
toMissingOfType(NA, type = "real")
toMissingOfType(NULL, type = "integer")
```

---

validateEnumValue *Check if* value *is in the given* enum*. If not, stops with an error.*

Description

Check if value is in the given enum. If not, stops with an error.

Usage

```
validateEnumValue(value, enum, nullAllowed = FALSE)
```

Arguments

value	A value to search for in the enum.
enum	enum where the value should be contained.
nullAllowed	If TRUE, value can be NULL and the test always passes. If FALSE (default), NULL is not accepted and the test fails.

Examples

```
Symbol <- enum(c(Diamond = 1, Triangle = 2, Circle = 2))
validateEnumValue(1, Symbol)
```

---

validateIsFileUTF8 *Validate if a file is UTF-8 encoded.*

Description

Validate if a file is UTF-8 encoded.

Usage

```
validateIsFileUTF8(file)
```

Arguments

file	A name of the file or full path.

Value

If validations are successful, validateIsFileUTF8() returns NULL. Otherwise, error is signaled.

Examples

```
writeLines(c("Hello, world!"), "utf.txt")
writeLines(c("Hello, world!", "\xb5g/L"), "non-utf.txt")

validateIsFileUTF8("utf.txt") # NULL
## Not run:
validateIsFileUTF8("non-utf.txt") # Error

## End(Not run)
```

`validateIsOfType` *Check if the provided object is of certain type. If not, stop with an error.*

Description

Check if the provided object is of certain type. If not, stop with an error.

Usage

```
validateIsOfType(object, type, nullAllowed = FALSE)

validateIsCharacter(object, nullAllowed = FALSE)

validateIsString(object, nullAllowed = FALSE)

validateIsNumeric(object, nullAllowed = FALSE)

validateIsInteger(object, nullAllowed = FALSE)

validateIsLogical(object, nullAllowed = FALSE)
```

Arguments

`object`	An object or an atomic vector or a list of objects.
`type`	A single string or a vector of string representation or class of the type that should be checked for.
`nullAllowed`	Boolean flag if `NULL` is accepted for the `object`. If `TRUE`, `NULL` always returns `TRUE`, otherwise `NULL` returns `FALSE`. Default is `FALSE`.

Value

`NULL` if the entered object is of expected type, otherwise produces error. Also accepts `NULL` as an input if `nullAllowed` argument is set to `TRUE`.

Examples

```
A <- data.frame(
  col1 = c(1, 2, 3),
  col2 = c(4, 5, 6),
  col3 = c(7, 8, 9)
)

validateIsOfType(A, "data.frame")
validateIsInteger(5)
validateIsNumeric(1.2)
validateIsCharacter("x")
validateIsLogical(TRUE)
```

---

validateIsOption *Validate Options Against Specifications*

Description

Validates a list of options against specified validation rules. Supports both modern spec constructors (`integerOption()`, etc.) and legacy list format for backward compatibility.

Usage

```
validateIsOption(options, validOptions)
```

Arguments

`options`	A list of options to validate.
`validOptions`	A list specifying validation rules for each option. Each entry should either be: • A spec object created with `integerOption()`, `characterOption()`, etc. • A list with fields: `type`, `valueRange`, `allowedValues`, `nullAllowed`, `naAllowed`

Details

Each entry in `validOptions` is validated against the matching value from `options`. Spec objects created with constructors (e.g., `integerOption()`) are recommended because they express intent clearly and work well with IDEs. For backward compatibility, legacy list-based specs are still accepted and are automatically normalized to `optionSpec` before validation.

Value

Returns `NULL` invisibly if all validations pass. Stops with detailed error message listing all failures if any validation fails.

Examples

```
validOptions <- list(
  maxIterations = integerOption(min = 1L, max = 10000L),
  method = characterOption(allowedValues = c("a", "b")),
  threshold = numericOption(min = 0, max = 1, nullAllowed = TRUE)
)

options <- list(maxIterations = 100L, method = "a", threshold = 0.05)
validateIsOption(options, validOptions)
```

---

validateVector *Validate Vector Against Specified Criteria*

Description

Validates a vector x based on specified criteria, including type correctness, value range, allowed values, and handling of NULL and NA values. If the vector fails any validation, an informative error message is thrown.

Usage

```
validateVector(
  x,
  type = NULL,
  valueRange = NULL,
  allowedValues = NULL,
  nullAllowed = FALSE,
  naAllowed = FALSE
)

validateVectorRange(x, type, valueRange)

validateVectorValues(x, type, allowedValues = NULL, naAllowed = FALSE)
```

Arguments

x	Vector to validate.
type	Expected type of elements in x ("numeric", "integer", "character", "factor", "logical", or "Date"). Type "double" is treated as "numeric".
valueRange	Optional vector of length 2 specifying the range of allowed values for x, applicable to "numeric", "integer", "character", and "Date" types.
allowedValues	Optional vector specifying a set of allowed values for x.
nullAllowed	Logical flag indicating whether x can be NULL. Defaults to FALSE.
naAllowed	Logical flag indicating whether elements in x can be NA. Defaults to FALSE.

Details

validateVector is the primary function for checking a vector against defined validation criteria. It ensures that x meets the type, range, and allowed value conditions specified. For more detailed validations related to the value range and allowed values, validateVectorRange and validateVectorValues functions are utilized respectively.

Value

Does not return a value explicitly but will stop with a descriptive error message if any of the validations fail.

Examples

```
validateVector(x = 1:5, type = "integer")
validateVector(x = c(1.2, 2.5), type = "numeric", valueRange = c(1, 3))
validateVector(x = c("a", "b"), type = "character", allowedValues = c("a", "b", "c"))
validateVector(
  x = as.Date("2020-01-01"), type = "Date",
  valueRange = as.Date(c("2020-01-01", "2020-12-31"))
)

# Range validation examples
validateVectorRange(x = c(5, 10), type = "numeric", valueRange = c(1, 10))
validateVectorRange(x = c("a", "b"), type = "character", valueRange = c("a", "c"))
validateVectorRange(
  x = as.Date(c("2020-01-01")), type = "Date",
  valueRange = as.Date(c("2020-01-01", "2020-12-31"))
)
validateVectorRange(x = 1:3, type = "integer", valueRange = c(1L, 5L))

# Allowed values validation examples
validateVectorValues(x = c("a", "b"), type = "character", allowedValues = c("a", "b", "c"))
validateVectorValues(x = c(2L, 4L), type = "integer", allowedValues = c(1L, 2L, 3L, 4L))
validateVectorValues(x = c(TRUE), type = "logical", allowedValues = c(TRUE, FALSE))
```

Index