Package 'BiocBaseUtils'

April 29, 2025

Title General utility functions for developing Bioconductor packages

Version 1.11.0

Description The package provides utility functions related to package development. These include functions that replace slots, and selectors for show methods. It aims to coalesce the various helper functions often re-used throughout the Bioconductor ecosystem.

Imports methods, utils

Depends R (>= 4.2.0)

Suggests knitr, rmarkdown, BiocStyle, tinytest

License Artistic-2.0

Encoding UTF-8

biocViews Software, Infrastructure

BugReports https://www.github.com/Bioconductor/BiocBaseUtils/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Date 2024-08-22

git_url https://git.bioconductor.org/packages/BiocBaseUtils

git_branch devel

git_last_commit b3d5af6

git_last_commit_date 2025-04-15

Repository Bioconductor 3.22

Date/Publication 2025-04-29

Author Marcel Ramos [aut, cre] (ORCID:
<https://orcid.org/0000-0002-3242-0582>),
Martin Morgan [ctb],
Hervé Pagès [ctb]

Maintainer Marcel Ramos <marcel.ramos@sph.cuny.edu>

Contents

BiocBaseUtils-package 2
askUserYesNo 2
Assertions 3
checkInstalled 4
lifeCycle 5
selectSome 6
setSlots 7

Index **9**

`BiocBaseUtils-package` *BiocBaseUtils: Utility and Internal functions for Bioconductor packages*

Description

BiocBaseUtils is a package aimed at helping the typical Bioconductor developer formalize often written functions that can be seen scattered throughout the Bioconductor ecosystem. Some of these functions include the ability to replace slots in an object. Other functions work to create a nice show method output by selecting some observations.

Author(s)

Maintainer: Marcel Ramos <marcel.ramos@sph.cuny.edu> (ORCID)

Other contributors:

- Martin Morgan <martin.morgan@roswellpark.org> [contributor]
- Hervé Pagès <hpages.on.github@gmail.com> [contributor]

See Also

Useful links:

- Report bugs at https://www.github.com/Bioconductor/BiocBaseUtils/issues

---

`askUserYesNo` *Ask user for a yes/no response*

Description

Ask user for a yes/no response

Usage

```
askUserYesNo(prompt, interactive.only = TRUE)
```

Arguments

prompt	character() Question form prompt to display to the user without a question mark
interactive.only	logical(1) If TRUE, the function will only prompt the user when the R session is interactive. If FALSE, the function will always prompt the user.

Value

TRUE when user replies with 'yes' to prompt, FALSE when 'no'

Author(s)

Martin M.

Examples

```
askUserYesNo("Do you want to continue")
```

Assertions *Suite of helper functions to test for types*

Description

These are a group of helper functions that allow the developer to easily check for common data types in Bioconductor. These include logical, character, and numeric (& integer).

Usage

```
isTRUEorFALSE(x, na.ok = FALSE)

isScalarCharacter(x, na.ok = FALSE, zchar = FALSE)

isScalarInteger(x, na.ok = FALSE)

isScalarNumber(x, na.ok = FALSE, infinite.ok = FALSE)

isScalarLogical(x, na.ok = FALSE)

isCharacter(x, na.ok = FALSE, zchar = FALSE)

isZeroOneCharacter(x, na.ok = FALSE, zchar = FALSE)
```

Arguments

x	The input vector whose type is to be checked
na.ok	logical(1L) Whether it is acceptable to consider NA type inputs (default: FALSE).
zchar	logical(1L) Whether is is acceptable to consider 'zero' characters as defined by nchar, e.g., nchar("") (default: FALSE).
infinite.ok	logical(1L) Whether it is acceptable to consider infinite values as identified by is.finite (default: FALSE).

Details

Some functions such as `isScalarCharacter` allow exceptions to the type checks via the `na.ok` and `zchar` arguments. Others, for example `isScalarNumber` can permit `Inf` with the `infinite.ok` argument.

Value

Either `TRUE` or `FALSE`

Functions

- `isTRUEorFALSE()`: Is the input a single logical vector?
- `isScalarCharacter()`: Is the input a single character vector?
- `isScalarInteger()`: Is the input a single integer vector?
- `isScalarNumber()`: Is the input a single numeric vector?
- `isScalarLogical()`: Is the input a single logical vector?
- `isCharacter()`: Is the input a character vector?
- `isZeroOneCharacter()`: Is the input a character vector of zero or one length?

Author(s)

M. Morgan, H. Pagès

Examples

```
isTRUEorFALSE(TRUE)
isTRUEorFALSE(FALSE)
isTRUEorFALSE(NA, na.ok = TRUE)

isScalarCharacter(LETTERS)
isScalarCharacter("L")
isCharacter(LETTERS)
isCharacter(NA_character_, na.ok = TRUE)
isZeroOneCharacter("")
isZeroOneCharacter("", zchar = TRUE)

isScalarInteger(1L)
isScalarInteger(1)

isScalarNumber(1)
isScalarNumber(1:2)
```

---

checkInstalled *Check packages are installed otherwise suggest*

Description

`checkInstalled` allows to check if a package is installed. If the package is not available, a convenient copy-and-paste message is provided for package installation with `BiocManager`. The function is typically used within functions that check for package availability from the `Suggests` field.

Usage

```
checkInstalled(pkgs)
```

Arguments

pkgs	character() package names required for a function

Value

TRUE if all packages are installed, otherwise stops with a message and suggests installation of missing packages

Author(s)

M. Morgan, M. Ramos

Examples

```
if (interactive()) {
    checkInstalled(
        c("BiocParallel", "SummarizedExperiment")
    )
}
```

---

lifeCycle *Set the life cycle stage of a function*

Description

The lifeCycle function is used to set the life cycle stage of a function. It is to be used within the body of the function that is being deprecated or defunct. It is a wrapper around both .Deprecated and .Defunct base R functions.

Usage

```
lifeCycle(
  newfun = oldfun,
  newpackage,
  package,
  cycle = c("deprecated", "defunct"),
  title = package
)
```

Arguments

newfun	character(1) The name of the function to use instead. It can be a specific function within another package (e.g., package::function) or a function in the current package (e.g., function). If newfun is not specified, the calling function oldfun is assumed to be the replacement.
newpackage	character(1) If a function is moved to a new package, the name of the new package can be specified here. This is equivalent to specifying newfun = paste0(newpackage, "::", newfun).

package	character(1) The name of the package where the deprecated or defunct function resides. It corresponds to the package from where the lifeCycle function is called.
cycle	character(1) The life cycle stage of the function. This can be either "deprecated" or "defunct".
title	character(1) The Rd name prefix of the documentation page where deprecated or defunct functions are documented (e.g., "summary" for "summary-deprecated"). By default, the package name is used.

Examples

```
test_fun <- function() {
    lifeCycle(newfun = "new_test", package = "BiocBaseUtils")
}
tryCatch(
   test_fun(),
   warning = function(w) message(w)
)
test_fun <- function() {
    lifeCycle(
        newfun = "new_test", package = "BiocBaseUtils", cycle = "defunct"
    )
}
tryCatch(
   test_fun(),
   error = function(e) message(e)
)
```

---

selectSome *Select and return only some entries from a vector*

Description

selectSome works well in show methods. It abbreviates a vector input depending on the maxToShow argument.

Usage

```
selectSome(
  obj,
  maxToShow = 5,
  ellipsis = "...",
  ellipsisPos = c("middle", "end", "start"),
  quote = FALSE
)
```

Arguments

obj	character() A vector to be abbreviated for display purposes
maxToShow	numeric(1) The maximum number of values to show in the output (default: 5)
ellipsis	character(1) The symbol used to abbreviate values in the vector (default: "...")

ellipsisPos	character(1) The location for the ellipsis in the output, by default in the "middle" but can be moved to either the "end" or the "start".
quote	logical(1) Whether or not to add a single quote around the obj input. This only works for character type inputs.

Value

An abbreviated output of obj

Author(s)

M. Morgan, H. Pagès

Examples

```
letters

selectSome(letters)
```

---

setSlots *Convenience function to set slot values*

Description

Given the current object, the function setSlots will take name-value pair inputs either as named arguments or a list and replace the values of the specified slots. This is a convenient function for updating slots in an S4 class object.

Usage

```
setSlots(object, ..., check = TRUE)
```

Arguments

object	An S4 object with slots to replace
...	Slot name and value pairs either as named arguments or a named list, e.g., slotName = value.
check	logical(1L) Whether to run validObject after the slot replacement

Value

The object input with updated slot data

Author(s)

H. Pagès

Examples

```
setClass("A", representation = representation(slotA = "character"))

aclass <- new("A", slotA = "A")

setSlots(aclass, slotA = "B")
```

Index

askUserYesNo, 2
Assertions, 3

BiocBaseUtils *(*BiocBaseUtils-package*)*, 2
BiocBaseUtils-package, 2

checkInstalled, 4

isCharacter *(*Assertions*)*, 3
isScalarCharacter *(*Assertions*)*, 3
isScalarInteger *(*Assertions*)*, 3
isScalarLogical *(*Assertions*)*, 3
isScalarNumber *(*Assertions*)*, 3
isTRUEorFALSE *(*Assertions*)*, 3
isZeroOneCharacter *(*Assertions*)*, 3

lifeCycle, 5

replaceSlots *(*setSlots*)*, 7

selectSome, 6
setSlots, 7