Package ‘BiocGenerics’

January 21, 2017

Title S4 generic functions for Bioconductor
Description S4 generic functions needed by many Bioconductor packages.
Version 0.20.0
Author The Bioconductor Dev Team
Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>
biocViews Infrastructure
Depends methods, utils, graphics, stats, parallel
Imports methods, utils, graphics, stats, parallel
Suggests Biobase, S4Vectors, IRanges, GenomicRanges, AnnotationDbi, oligoClasses, oligo, affyPLM, flowClust, affy, DESeq2, MSnbase, annotate, RUnit
License Artistic-2.0

NeedsCompilation no

R topics documented:

BiocGenerics-package .......................................................... 3
annotation ................................................................. 5
append ................................................................. 6
as.data.frame .......................................................... 7
as.vector ............................................................... 8
boxplot ................................................................. 9
cbind ................................................................. 10
clusterApply ......................................................... 11
combine ............................................................. 13
R topics documented:

dbconn .......................................................... 15
density .......................................................... 16
dge ................................................................. 17
do.call ........................................................... 17
duplicated ......................................................... 18
eval ................................................................. 20
evalq .............................................................. 21
Extremes .......................................................... 21
fileName .......................................................... 23
funprog ............................................................ 23
get ................................................................. 25
grep ............................................................... 26
image .............................................................. 27
IQR ................................................................. 28
is.unsorted ....................................................... 28
lapply ............................................................. 30
lengths ............................................................ 31
mad ................................................................. 32
mapply ............................................................ 33
match ............................................................. 34
normalize ......................................................... 35
nrow ............................................................... 36
order .............................................................. 37
organism_species .............................................. 38
paste .............................................................. 39
plotMA ........................................................... 40
plotPCA .......................................................... 41
rank ............................................................... 42
relist ............................................................. 43
rep ................................................................. 44
residuals .......................................................... 45
row+colnames ................................................... 45
S3-classes-as-S4-classes ....................................... 47
score .............................................................. 47
sets ............................................................... 48
sort ............................................................... 49
start .............................................................. 50
strand ............................................................. 52
subset ............................................................ 53
table .............................................................. 54
tapply ............................................................. 55
testPackage ...................................................... 56
unique ........................................................... 57
unlist ............................................................. 58
unsplit ........................................................... 59
updateObject ..................................................... 60
weights .......................................................... 62
which ............................................................. 63
xtabs ............................................................. 64

Index ............................................................ 66
Description

S4 generic functions needed by many Bioconductor packages.

Details

We divide the generic functions defined in the BiocGenerics package in 2 categories: (1) functions already defined in base R and explicitly promoted to generics in BiocGenerics, and (2) Bioconductor specific generics.

(1) Functions defined in base R and explicitly promoted to generics in the BiocGenerics package:
Generics for functions defined in package base:

- BiocGenerics::append
- BiocGenerics::as.data.frame
- BiocGenerics::as.vector
- BiocGenerics::cbind, BiocGenerics::rbind
- BiocGenerics::do.call
- BiocGenerics::duplicated, BiocGenerics::anyDuplicated
- BiocGenerics::eval
- Extremes: BiocGenerics::pmax, BiocGenerics::pmin, BiocGenerics::pmax.int, BiocGenerics::pmin.int
- funprog: BiocGenerics::Reduce, BiocGenerics::Filter, BiocGenerics::Find, BiocGenerics::Map, BiocGenerics::Position
- BiocGenerics::get, BiocGenerics::mget
- BiocGenerics::grep, BiocGenerics::grepl
- BiocGenerics::is.unsorted
- BiocGenerics::lapply, BiocGenerics::sapply
- BiocGenerics::lengths
- BiocGenerics::mapply
- BiocGenerics::match
- BiocGenerics::nrow, BiocGenerics::ncol, BiocGenerics::NROW, BiocGenerics::NCOL
- BiocGenerics::order
- BiocGenerics::paste
- BiocGenerics::rank
- BiocGenerics::rep.int
- BiocGenerics::rownames, BiocGenerics::rownames<-, BiocGenerics::colnames, BiocGenerics::colnames<-
- sets: BiocGenerics::union, BiocGenerics::intersect, BiocGenerics::setdiff
- BiocGenerics::sort
- BiocGenerics::start, BiocGenerics::start<-, BiocGenerics::end, BiocGenerics::end<-
- BiocGenerics::width, BiocGenerics::width<-
- BiocGenerics::subset
- BiocGenerics::table
- BiocGenerics::tapply
Generics for functions defined in package `stats`:
- `BiocGenerics::density`
- `BiocGenerics::residuals`
- `BiocGenerics::weights`
- `BiocGenerics::xtabs`

Generics for functions defined in package `parallel`:
- `BiocGenerics::clusterCall`
- `BiocGenerics::clusterApply`
- `BiocGenerics::clusterApplyLB`
- `BiocGenerics::clusterEvalQ`
- `BiocGenerics::clusterExport`
- `BiocGenerics::clusterMap`
- `BiocGenerics::clusterSplit`
- `BiocGenerics::parLapply`
- `BiocGenerics::parSapply`
- `BiocGenerics::parApply`
- `BiocGenerics::parRapply`
- `BiocGenerics::parCapply`
- `BiocGenerics::parLapplyLB`
- `BiocGenerics::parSapplyLB`

(2) Bioconductor specific generics:
- `annotation`
- `annotation<-`
- `combine`
- `dbconn, dbfile`
- `fileName`
- `normalize`
- `organism, organism<-`, `species, species<-`
- `plotMA`
- `plotPCA`
- `score, score<-`
- `strand, strand<-`, `invertStrand`
- `updateObject`

**Note**

More generics can be added on request by sending an email to the Bioc-devel mailing list: [http://bioconductor.org/help/mailing-list/](http://bioconductor.org/help/mailing-list/)

Things that should NOT be added to the `BiocGenerics` package:

- Internal generic primitive functions like `length, dim, `dim<-`, etc... See `?InternalMethods` for the complete list. There are a few exceptions though, that is, the `BiocGenerics` package may actually redefine a few of those internal generic primitive functions as S4 generics when for example the signature of the internal generic primitive is not appropriate (this is the case for `BiocGenerics::cbind`).
- S3 and S4 group generic functions like `Math, Ops`, etc... See `?groupGeneric` and `?S4groupGeneric` for the complete list.
- Generics already defined in the `stats4` package.
Author(s)

The Bioconductor Dev Team

See Also

- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `setGeneric` and `setMethod` for defining generics and methods.

Examples

```r
## List all the symbols defined in this package:
ls('package:BiocGenerics')
```

---

### Accessing annotation information

Description

Get or set the annotation information contained in an object.

Usage

```r
annotation(object, ...) annotation(object, ...) <- value
```

Arguments

- `object` An object containing annotation information.
- `...` Additional arguments, for use in specific methods.
- `value` The annotation information to set on `object`.

See Also

- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `annotation.eSet-method` in the `Biobase` package for an example of a specific annotation method (defined for `eSet` objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

```r
annotation
showMethods("annotation")

library(Biobase)
showMethods("annotation")
selectMethod("annotation", "eSet")
```
append

Append elements to a vector-like object

Description

Append (or insert) elements to (in) a vector-like object.

NOTE: This man page is for the `append` S4 generic function defined in the `BiocGenerics` package. See `?base::append` for the default method (defined in the `base` package). Bioconductor packages can define specific methods for objects (typically vector-like or data-frame-like) not supported by the default method.

Usage

`append(x, values, after=length(x))`

Arguments

- `x`: The vector-like object to be modified.
- `values`: The vector-like object containing the values to be appended to `x`. `values` would typically be of the same class as `x`, but not necessarily.
- `after`: A subscript, after which the values are to be appended.

Value

See `?base::append` for the value returned by the default method.

Specific methods defined in Bioconductor packages will typically return an object of the same class as `x` and of length `length(x) + length(values)`.

See Also

- `base::append` for the default append method.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `append,Vector,Vector-method` in the `S4Vectors` package for an example of a specific append method (defined for `Vector` objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

```r
append # note the dispatch on the 'x' and 'values' args only
showMethods("append")
selectMethod("append", c("ANY", "ANY")) # the default method
```
**as.data.frame**  

Coerce an object into a data frame

---

**Description**

Function to coerce to a data frame, if possible.

NOTE: This man page is for the `as.data.frame` S4 generic function defined in the **BiocGenerics** package. See `?base::as.data.frame` for the default method (defined in the **base** package). Bioconductor packages can define specific methods for objects not supported by the default method.

**Usage**

```r
as.data.frame(x, row.names=NULL, optional=FALSE, ...)
```

**Arguments**

- `x`  
The object to coerce.
- `row.names`, `optional`, `...`  
  See `?base::as.data.frame` for a description of these arguments.

**Value**

A data frame.

See `?base::as.data.frame` for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

**See Also**

- `base::as.data.frame` for the default `as.data.frame` method.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.

- `as.data.frame,DataFrame-method` in the **S4Vectors** package, and `as.data.frame,Ranges-method` in the **IRanges** package, for examples of specific `as.data.frame` methods (defined for `DataFrame` and `Ranges` objects, respectively).

- **BiocGenerics** for a summary of all the generics defined in the **BiocGenerics** package.

**Examples**

```r
as.data.frame  # note the dispatch on the 'x' arg only
showMethods("as.data.frame")
selectMethod("as.data.frame", "ANY")  # the default method
```
**as.vector**

Coerce an object into a vector

**Description**

Attempt to coerce an object into a vector of the specified mode. If the mode is not specified, attempt to coerce to whichever vector mode is considered more appropriate for the class of the supplied object.

NOTE: This man page is for the as.vector S4 generic function defined in the Bioconductor package. See ?base::as.vector for the default method (defined in the base package). Bioconductor packages can define specific methods for objects not supported by the default method.

**Usage**

as.vector(x, mode="any")

**Arguments**

- **x** The object to coerce.
- **mode** See ?base::as.vector for a description of this argument.

**Value**

A vector.

See ?base::as.vector for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

**See Also**

- base::as.vector for the default as.vector method.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- as.vector,Rle-method and as.vector,AtomicList-method in the S4Vectors and IRanges packages, respectively, for examples of specific as.vector methods (defined for Rle and AtomicList objects, respectively).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

**Examples**

as.vector # note the dispatch on the 'x' arg only
globals::showMethods("as.vector")
globals::selectMethod("as.vector", "ANY") # the default method
Description

Produce box-and-whisker plot(s) of the given (grouped) values.

NOTE: This man page is for the boxplot S4 generic function defined in the BiocGenerics package. See ?graphics::boxplot for the default method (defined in the graphics package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

boxplot(x, ...)

Arguments

x, ... See ?graphics::boxplot.

Value

See ?graphics::boxplot for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

See Also

• graphics::boxplot for the default boxplot method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• boxplot.FeatureSet-method in the oligo package for an example of a specific boxplot method (defined for FeatureSet objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

boxplot
showMethods("boxplot")
selectMethod("boxplot", "ANY") # the default method
**cbind**  
*Combine objects by rows or columns*

Description

cbind and rbind take one or more objects and combine them by columns or rows, respectively.

NOTE: This man page is for the cbind and rbind S4 generic functions defined in the **BiocGenerics** package. See ?base::cbind for the default methods (defined in the **base** package). Bioconductor packages can define specific methods for objects (typically vector-like or matrix-like) not supported by the default methods.

Usage

cbind(..., deparse.level=1)  
rbind(..., deparse.level=1)

Arguments

... One or more vector-like or matrix-like objects. These can be given as named arguments.

deparse.level See ?base::cbind for a description of this argument.

Value

See ?base::cbind for the value returned by the default methods.

Specific methods defined in Bioconductor packages will typically return an object of the same class as the input objects.

See Also

- base::cbind for the default cbind and rbind methods.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- cbind.DataFrame-method in the **S4Vectors** package for an example of a specific cbind method (defined for DataFrame objects).
- BiocGenerics for a summary of all the generics defined in the **BiocGenerics** package.

Examples

cbind  # note the dispatch on the '...' arg only  
showMethods("cbind")  
selectMethod("cbind", "ANY")  # the default method

rbind  # note the dispatch on the '...' arg only  
showMethods("rbind")  
selectMethod("rbind", "ANY")  # the default method
clusterApply

Apply operations using clusters

Description

These functions provide several ways to parallelize computations using a cluster.

NOTE: This man page is for the clusterCall, clusterApply, clusterApplyLB, clusterEvalQ, clusterExport, clusterMap, clusterSplit, parLapply, parSapply, parApply, parRapply, parCapply, parLapplyLB, and parSapplyLB S4 generic functions defined in the BiocGenerics package. See ?parallel::clusterApply for the default methods (defined in the parallel package). Bioconductor packages can define specific methods for cluster-like objects not supported by the default methods.

Usage

clusterCall(cl=NULL, fun, ...)
clusterApply(cl=NULL, x, fun, ...)
clusterApplyLB(cl=NULL, x, fun, ...)
clusterEvalQ(cl=NULL, expr)
clusterExport(cl=NULL, varlist, envir=.GlobalEnv)
clusterMap(cl=NULL, fun, ..., MoreArgs=NULL, RECYCLE=TRUE,
SIMPLIFY=FALSE, USE.NAMES=TRUE,
.scheduling=c("static", "dynamic"))
clusterSplit(cl=NULL, seq)
parLapply(cl=NULL, X, fun, ...)
parSapply(cl=NULL, X, FUN, ..., simplify=TRUE, USE.NAMES=TRUE)
parApply(cl=NULL, X, MARGIN, FUN, ...)
parRapply(cl=NULL, x, FUN, ...)
parCapply(cl=NULL, x, FUN, ...)
parLapplyLB(cl=NULL, X, fun, ...)
parSapplyLB(cl=NULL, X, FUN, ..., simplify=TRUE, USE.NAMES=TRUE)

Arguments

c1 A cluster-like object.
x A vector-like object for clusterApply and clusterApplyLB. A matrix-like object for parRapply and parCapply.
seq Vector-like object to split.
X A vector-like object for parLapply, parSapply, parLapplyLB, and parSapplyLB. An array-like object for parApply.
fun, ..., expr, varlist, envir, MoreArgs, RECYCLE, SIMPLIFY, USE.NAMES, .scheduling, FUN, simplify

See ?parallel::clusterApply for a description of these arguments.

Value

See ?parallel::clusterApply for the value returned by the default methods.

Specific methods defined in Bioconductor packages should behave like the default methods.
See Also

- `parallel::clusterApply` for the default methods.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

clusterCall # note the dispatch on the 'cl' arg only
showMethods("clusterCall")
selectMethod("clusterCall", "ANY") # the default method

clusterApply # note the dispatch on the 'cl' and 'x' args only
showMethods("clusterApply")
selectMethod("clusterApply", c("ANY", "ANY")) # the default method

clusterApplyLB # note the dispatch on the 'cl' and 'x' args only
showMethods("clusterApplyLB")
selectMethod("clusterApplyLB", c("ANY", "ANY")) # the default method

clusterEvalQ # note the dispatch on the 'cl' arg only
showMethods("clusterEvalQ")
selectMethod("clusterEvalQ", "ANY") # the default method

clusterExport # note the dispatch on the 'cl' arg only
showMethods("clusterExport")
selectMethod("clusterExport", "ANY") # the default method

clusterMap # note the dispatch on the 'cl' arg only
showMethods("clusterMap")
selectMethod("clusterMap", "ANY") # the default method

clusterSplit
showMethods("clusterSplit")
selectMethod("clusterSplit", c("ANY", "ANY")) # the default method

parLapply # note the dispatch on the 'cl' and 'X' args only
showMethods("parLapply")
selectMethod("parLapply", c("ANY", "ANY")) # the default method

parSapply # note the dispatch on the 'cl' and 'X' args only
showMethods("parSapply")
selectMethod("parSapply", c("ANY", "ANY")) # the default method

parApply # note the dispatch on the 'cl' and 'X' args only
showMethods("parApply")
selectMethod("parApply", c("ANY", "ANY")) # the default method

parRapply # note the dispatch on the 'cl' and 'x' args only
showMethods("parRapply")
selectMethod("parRapply", c("ANY", "ANY")) # the default method

parCapply # note the dispatch on the 'cl' and 'x' args only
showMethods("parCapply")
selectMethod("parCapply", c("ANY", "ANY")) # the default method
The `combine` generic function handles methods for combining or merging different Bioconductor data structures. It should, given an arbitrary number of arguments of the same class (possibly by inheritance), combine them into a single instance in a sensible way (some methods may only combine 2 objects, ignoring ... in the argument list; because Bioconductor data structures are complicated, check carefully that `combine` does as you intend).

**Usage**

```r
combine(x, y, ...)  
```

**Arguments**

- `x` One of the values.
- `y` A second value.
- `...` Any other objects of the same class as `x` and `y`.

**Details**

There are two basic combine strategies. One is an intersection strategy. The returned value should only have rows (or columns) that are found in all input data objects. The union strategy says that the return value will have all rows (or columns) found in any one of the input data objects (in which case some indication of what to use for missing values will need to be provided).

These functions and methods are currently under construction. Please let us know if there are features that you require.

**Value**

A single value of the same class as the most specific common ancestor (in class terms) of the input values. This will contain the appropriate combination of the data in the input values.

**Methods**

The following methods are defined in the `BiocGenerics` package:

- `combine(x=ANY, missing)` Return the first (`x`) argument unchanged.
combine(data.frame, data.frame) Combines two data.frame objects so that the resulting data.frame contains all rows and columns of the original objects. Rows and columns in the returned value are unique, that is, a row or column represented in both arguments is represented only once in the result. To perform this operation, combine makes sure that data in shared rows and columns are identical in the two dataframes. Data differences in shared rows and columns usually cause an error. combine issues a warning when a column is a factor and the levels of the factor in the two dataframes are different.

combine(matrix, matrix) Combined two matrix objects so that the resulting matrix contains all rows and columns of the original objects. Both matrices must have dimnames. Rows and columns in the returned value are unique, that is, a row or column represented in both arguments is represented only once in the result. To perform this operation, combine makes sure that data in shared rows and columns are all equal in the two matrices.

Additional combine methods are defined in the Biobase package for AnnotatedDataFrame, AssayData, MIAME, and eSet objects.

Author(s)
Biocore

See Also
• merge for merging two data frames (or data.frame-like) objects.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples
combine
showMethods("combine")
selectMethod("combine", c("ANY", "missing"))
selectMethod("combine", c("data.frame", "data.frame"))
selectMethod("combine", c("matrix", "matrix"))

## ----------------------------------------
## COMBINING TWO DATA FRAMES
## ----------------------------------------
x <- data.frame(x=1:5,
y=factor(letters[1:5], levels=letters[1:8]),
row.names=letters[1:5])
y <- data.frame(z=3:7,
y=factor(letters[3:7], levels=letters[1:8]),
row.names=letters[3:7])
combine(x, y)

w <- data.frame(w=4:8,
y=factor(letters[4:8], levels=letters[1:8]),
row.names=letters[4:8])
combine(w, x, y)
# y is converted to 'factor' with different levels
df1 <- data.frame(x=1:5, y=letters[1:5], row.names=letters[1:5])
df2 <- data.frame(z=3:7, y=letters[3:7], row.names=letters[3:7])
try(combine(df1, df2)) # fails

# solution 1: ensure identical levels
y1 <- factor(letters[1:5], levels=letters[1:7])
y2 <- factor(letters[3:7], levels=letters[1:7])
df1 <- data.frame(x=1:5, y=y1, row.names=letters[1:5])
df2 <- data.frame(z=3:7, y=y2, row.names=letters[3:7])
combine(df1, df2)

# solution 2: force column to be 'character'
df1 <- data.frame(x=1:5, y=I(letters[1:5]), row.names=letters[1:5])
df2 <- data.frame(z=3:7, y=I(letters[3:7]), row.names=letters[3:7])
combine(df1, df2)

## ---------------------------------------------------------------------
## COMBINING TWO MATRICES
## ---------------------------------------------------------------------
m <- matrix(1:20, nrow=5, dimnames=list(LETTERS[1:5], letters[1:4]))
combine(m[1:3,], m[4:5,])
combine(m[1:3, 1:3], m[3:5, 3:4]) # overlap

---

### dbconn

Accessing SQLite DB information

**Description**

Get a connection object or file path for a SQLite DB

**Usage**

dbconn(x)
dbfile(x)

**Arguments**

- **x** An object with a SQLite connection.

**Value**

dbconn returns a connection object to the SQLite DB containing x’s data.
dbfile returns a path (character string) to the SQLite DB (file) containing x’s data.

**See Also**

- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `dbconn, AnnotationDb-method` in the `AnnotationDbi` package for an example of a specific dbconn method (defined for dbconn objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.
Examples

dbconn
  showMethods("dbconn")
dbfile
  showMethods("dbfile")

library(AnnotationDbi)
  showMethods("dbconn")
  selectMethod("dbconn", "AnnotationDb")

density kernel density estimation

Description

The generic function `density` computes kernel density estimates.

NOTE: This man page is for the `density` S4 generic function defined in the BiocGenerics package. See `?stats::density` for the default method (defined in the stats package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

density(x, ...)

Arguments

  x, ... See `?stats::density`.

Value

See `?stats::density` for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

See Also

  - `stats::density` for the default density method.
  - `showMethods` for displaying a summary of the methods defined for a given generic function.
  - `selectMethod` for getting the definition of a specific method.
  - `density.flowClust-method` in the flowClust package for an example of a specific density method (defined for flowClust objects).
  - BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

density
  showMethods("density")
  selectMethod("density", "ANY") # the default method
**Accessors and generic functions used in the context of count datasets**

**Description**

These generic functions provide basic interfaces to operations on and data access to count datasets.

**Usage**

- `counts(object, ...)`
- `counts(object, ...) <- value`
- `dispTable(object, ...)`
- `dispTable(object, ...) <- value`
- `sizeFactors(object, ...)`
- `sizeFactors(object, ...) <- value`
- `conditions(object, ...)`
- `conditions(object, ...) <- value`
- `design(object, ...)`
- `design(object, ...) <- value`
- `estimateSizeFactors(object, ...)`
- `estimateDispersions(object, ...)`
- `plotDispEsts(object, ...)`

**Arguments**

- **object**: Object of class for which methods are defined, e.g., `CountDataSet`, `DESeqSummarizedExperiment`, or `ExonCountSet`.
- **value**: Value to be assigned to corresponding components of `object`; supported types depend on method implementation.
- **...**: Further arguments, perhaps used by methods.

**Details**

For the details, please consult the manual pages of the methods in the `DESeq`, `DESeq2`, and `DEXSeq` packages and the package vignettes.

**Author(s)**

W. Huber, S. Anders

---

**do.call**

*Execute a function call*

**Description**

`do.call` constructs and executes a function call from a name or a function and a list of arguments to be passed to it.

*NOTE:* This man page is for the `do.call S4 generic function` defined in the `BiocGenerics` package. See `?base::do.call` for the default method (defined in the `base` package). Bioconductor packages can define specific methods for objects not supported by the default method.
Usage

do.call(what, args, quote=FALSE, envir=parent.frame())

Arguments

what  The default method expects either a function or a non-empty character string naming the function to be called. See ?base::do.call for the details. Specific methods can support other objects. Please refer to the documentation of a particular method for the details.

args  The default method expects a list of arguments to the function call (the names attribute of args gives the argument names). See ?base::do.call for the details. Specific methods can support other objects. Please refer to the documentation of a particular method for the details.

quote, envir  See ?base::do.call for a description of these arguments.

Value

The result of the (evaluated) function call.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

See Also

• base::do.call for the default do.call method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

do.call # note the dispatch on the 'what' and 'args' args only
showMethods("do.call")
selectMethod("do.call", c("ANY", "ANY"))  # the default method

duplicated  Determine duplicate elements

Description

Determines which elements of a vector-like or data-frame-like object are duplicates of elements with smaller subscripts, and returns a logical vector indicating which elements (rows) are duplicates.

NOTE: This man page is for the duplicated and anyDuplicated S4 generic functions defined in the BiocGenerics package. See ?base::duplicated for the default methods (defined in the base package). Bioconductor packages can define specific methods for objects (typically vector-like or data-frame-like) not supported by the default method.
duplication

Usage

duplicated(x, incomparables=FALSE, ...)
anyDuplicated(x, incomparables=FALSE, ...)

Arguments

x
A vector-like or data-frame-like object.
incomparables, ...
See ?base::duplicated for a description of these arguments.

Value

The default duplicated method (see ?base::duplicated) returns a logical vector of length N where N is:

- length(x) when x is a vector;
- nrow(x) when x is a data frame.

Specific duplicated methods defined in Bioconductor packages must also return a logical vector of the same length as x when x is a vector-like object, and a logical vector with one element for each row when x is a data-frame-like object.

The default anyDuplicated method (see ?base::duplicated) returns a single non-negative integer and so must the specific anyDuplicated methods defined in Bioconductor packages.

anyDuplicated should always behave consistently with duplicated.

See Also

- base::duplicated for the default duplicated and anyDuplicated methods.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- duplicated.Rle-method in the S4Vectors package for an example of a specific duplicated method (defined for Rle objects).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

duplicated
showMethods("duplicated")
selectMethod("duplicated", "ANY") # the default method

anyDuplicated
showMethods("anyDuplicated")
selectMethod("anyDuplicated", "ANY") # the default method
eval

Evaluate an (unevaluated) expression

Description

eval evaluates an R expression in a specified environment.

NOTE: This man page is for the eval S4 generic function defined in the BiocGenerics package. See ?base::eval for the default method (defined in the base package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

eval(expr, envir=parent.frame(),
    enclos=if (is.list(envir) || is.pairlist(envir))
    parent.frame() else baseenv())

Arguments

expr
An object to be evaluated. May be any object supported by the default method (see ?base::eval) or by the additional methods defined in Bioconductor packages.

envir
The environment in which expr is to be evaluated. May be any object supported by the default method (see ?base::eval) or by the additional methods defined in Bioconductor packages.

enclos
See ?base::eval for a description of this argument.

Value

See ?base::eval for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

See Also

• base::eval for the default eval method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• eval,expression,Vector-method in the IRanges package for an example of a specific eval method (defined for when the expr and envir arguments are an expression and a Vector object, respectively).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

eval # note the dispatch on 'expr' and 'envir' args only
showMethods("eval")
selectMethod("eval", c("ANY", "ANY")) # the default method
evalq

Description

evalq evaluates an R expression (the quoted form of its first argument) in a specified environment.

NOTE: This man page is for the evalq wrapper defined in the BiocGenerics package. See ?base::evalq for the function defined in the base package. This wrapper correctly delegates to the eval generic, rather than base::eval.

Usage

```r
evalq(expr, envir=parent.frame(),
  enclos=if (is.list(envir) || is.pairlist(envir))
    parent.frame() else baseenv())
```

Arguments

- `expr`  
  Quoted to form the expression that is evaluated.
- `envir`  
  The environment in which expr is to be evaluated. May be any object supported by methods on the eval generic.
- `enclos`  
  See ?base::evalq for a description of this argument.

Value

See ?base::evalq.

See Also

- base::evalq for the base evalq function.

Examples

```r
evalq # note just a copy of the original evalq
```

Extremes

Maxima and minima

Description

pmax, pmin, pmax.int and pmin.int return the parallel maxima and minima of the input values.

NOTE: This man page is for the pmax, pmin, pmax.int and pmin.int S4 generic functions defined in the BiocGenerics package. See ?base::pmax for the default methods (defined in the base package). Bioconductor packages can define specific methods for objects (typically vector-like or matrix-like) not supported by the default methods.
Usage

\texttt{pmax(..., na.rm=FALSE)}
\texttt{pmin(..., na.rm=FALSE)}
\texttt{pmax.int(..., na.rm=FALSE)}
\texttt{pmin.int(..., na.rm=FALSE)}

Arguments

... \text{One or more vector-like or matrix-like objects.}
\texttt{na.rm} \text{ See \texttt{?base::pmax} for a description of this argument.}

Value

See \texttt{?base::pmax} for the value returned by the default methods.

Specific methods defined in Bioconductor packages will typically return an object of the same class as the input objects.

See Also

• \texttt{base::pmax} for the default \texttt{pmax}, \texttt{pmin}, \texttt{pmax.int} and \texttt{pmin.int} methods.
• \texttt{showMethods} for displaying a summary of the methods defined for a given generic function.
• \texttt{selectMethod} for getting the definition of a specific method.
• \texttt{pmax.Rle-method} in the \texttt{S4Vectors} package for an example of a specific \texttt{pmax} method (defined for \texttt{Rle} objects).
• \texttt{BiocGenerics} for a summary of all the generics defined in the \texttt{BiocGenerics} package.

Examples

\texttt{pmax}
\texttt{showMethods("pmax")}
\texttt{selectMethod("pmax", "ANY") \# the default method}

\texttt{pmin}
\texttt{showMethods("pmin")}
\texttt{selectMethod("pmin", "ANY") \# the default method}

\texttt{pmax.int}
\texttt{showMethods("pmax.int")}
\texttt{selectMethod("pmax.int", "ANY") \# the default method}

\texttt{pmin.int}
\texttt{showMethods("pmin.int")}
\texttt{selectMethod("pmin.int", "ANY") \# the default method}
fileName

Accessing the file name of an object

Description

Get the file name of an object.

Usage

fileName(object, ...)

Arguments

object An object with a file name.
... Additional arguments, for use in specific methods.

See Also

- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- fileName.MSmap-method in the MSnbase package for an example of a specific fileName method (defined for MSmap objects).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

fileName
showMethods("fileName")

library(MSnbase)
showMethods("fileName")
selectMethod("fileName", "MSmap")

funprog

Common higher-order functions in functional programming languages

Description

Reduce uses a binary function to successively combine the elements of a given list-like or vector-like object and a possibly given initial value. Filter extracts the elements of a list-like or vector-like object for which a predicate (logical) function gives true. Find and Position give the first or last such element and its position in the object, respectively. Map applies a function to the corresponding elements of given list-like or vector-like objects.

NOTE: This man page is for the Reduce, Filter, Find, Map and Position S4 generic functions defined in the BiocGenerics package. See ?base::Reduce for the default methods (defined in the base package). Bioconductor packages can define specific methods for objects (typically list-like or vector-like) not supported by the default methods.
Usage
Reduce(f, x, init, right=FALSE, accumulate=FALSE)
Filter(f, x)
Find(f, x, right=FALSE, nomatch=NULL)
Map(f, ...)
Position(f, x, right=FALSE, nomatch=NA_integer_)

Arguments
f, init, right, accumulate, nomatch
  See ?base::Reduce for a description of these arguments.
x
  A list-like or vector-like object.
...
  One or more list-like or vector-like objects.

Value
See ?base::Reduce for the value returned by the default methods.
Specific methods defined in Bioconductor packages should behave as consistently as possible with the default methods.

See Also
• base::Reduce for the default Reduce, Filter, Find, Map and Position methods.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• Reduce.List-method in the S4Vectors package for an example of a specific Reduce method (defined for List objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples
Reduce # note the dispatch on the 'x' arg only
showMethods("Reduce")
selectMethod("Reduce", "ANY") # the default method

Filter # note the dispatch on the 'x' arg only
showMethods("Filter")
selectMethod("Filter", "ANY") # the default method

Find # note the dispatch on the 'x' arg only
showMethods("Find")
selectMethod("Find", "ANY") # the default method

Map # note the dispatch on the '...' arg only
showMethods("Map")
selectMethod("Map", "ANY") # the default method

Position # note the dispatch on the 'x' arg only
showMethods("Position")
selectMethod("Position", "ANY") # the default method
get

Return the value of a named object

Description

Search for an object with a given name and return it.

NOTE: This man page is for the get and mget S4 generic functions defined in the BiocGenerics package. See ?base::get for the default methods (defined in the base package). Bioconductor packages can define specific methods for objects (list-like or environment-like) not supported by the default methods.

Usage

\[
\begin{align*}
\text{get}(x, \text{pos}=-1, \text{envir}=\text{as.environment}(\text{pos}), \text{mode}=\text{"any"}, \text{inherits}=\text{TRUE}) \\
\text{mget}(x, \text{envir}, \text{mode}=\text{"any"}, \text{ifnotfound}, \text{inherits}=\text{FALSE})
\end{align*}
\]

Arguments

- **x**: For get: A variable name (or, more generally speaking, a key), given as a single string. For mget: A vector of variable names (or keys).
- **envir**: Where to look for the key(s). Typically a list-like or environment-like object.
- **pos**, **mode**, **inherits**, **ifnotfound**: See ?base::get for a description of these arguments.

Details

See ?base::get for details about the default methods.

Value

For get: The value corresponding to the specified key.

For mget: The list of values corresponding to the specified keys. The returned list must have one element per key, and in the same order as in x.

See ?base::get for the value returned by the default methods.

See Also

- base::get for the default get and mget methods.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- get,ANY,Bimap,missing-method in the AnnotationDbi package for an example of a specific get method (defined for Bimap objects).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.
Examples

```r
grep # note the dispatch on the 'x', 'pos' and 'envir' args only
showMethods("grep")
selectMethod("grep", c("ANY", "ANY", "ANY")) # the default method

mget # note the dispatch on the 'x' and 'envir' args only
showMethods("mget")
selectMethod("mget", c("ANY", "ANY")) # the default method
```

grep

**Pattern Matching and Replacement**

**Description**

Search for matches to argument ‘pattern’ within each element of a character vector.

NOTE: This man page is for the `grep` and `grepl` S4 generic functions defined in the `BiocGenerics` package. See `?base::grep` for the default methods (defined in the `base` package). Bioconductor packages can define specific methods for objects not supported by the default method.

**Usage**

```r
grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
     fixed = FALSE, useBytes = FALSE, invert = FALSE)
grepl(pattern, x, ignore.case = FALSE, perl = FALSE,
      fixed = FALSE, useBytes = FALSE)
```

**Arguments**

- `pattern` The pattern for searching in `x`, such as a regular expression.
- `x` The character vector (in the general sense) to search.
- `ignore.case`, `perl`, `value`, `fixed`, `useBytes`, `invert`
  See `?base::grep` for a description of these arguments.

**Value**

See `?base::grep` for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

**See Also**

- `base::grep` for the default grep and grepl methods.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

**Examples**

```r
grep # note the dispatch on 'pattern' and 'x' args only
showMethods("grep")
selectMethod("grep", "ANY") # the default method
```
Display a color image

Description

Creates a grid of colored or gray-scale rectangles with colors corresponding to the values in z. This can be used to display three-dimensional or spatial data aka *images.*

NOTE: This man page is for the image S4 generic function defined in the BiocGenerics package. See ?graphics::image for the default method (defined in the graphics package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

image(x, ...)

Arguments

x, ... See ?graphics::image.

Details

See ?graphics::image for the details.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

See Also

- graphics::image for the default image method.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- image,FeatureSet-method in the oligo package for an example of a specific image method (defined for FeatureSet objects).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

image
showMethods("image")
selectMethod("image", "ANY") # the default method
IQR

The Interquartile Range

Description

Compute the interquartile range for a vector.

NOTE: This man page is for the IQR S4 generic function defined in the BiocGenerics package. See ?stats::IQR for the default method (defined in the stats package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

IQR(x, na.rm = FALSE, type = 7)

Arguments

x, na.rm, type See ?stats::IQR.

Value

See ?stats::IQR for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

See Also

• stats::IQR for the default IQR method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

IQR
showMethods("IQR")
selectMethod("IQR", "ANY") # the default method

is.unsorted

Test if a vector-like object is not sorted

Description

Test if a vector-like object is not sorted, without the cost of sorting it.

NOTE: This man page is for the is.unsorted S4 generic function defined in the BiocGenerics package. See ?base::is.unsorted for the default method (defined in the base package). Bioconductor packages can define specific methods for objects (typically vector-like) not supported by the default method.
is.unsorted

Usage

is.unsorted(x, na.rm=FALSE, strictly=FALSE, ...)

Arguments

x A vector-like object.

na.rm, strictly

See ?base::is.unsorted for a description of these arguments.

... Additional arguments, for use in specific methods.

Note that base::is.unsorted (the default method) only takes the x, na.rm, and strictly arguments.

Value

See ?base::is.unsorted for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

Note

TO DEVELOPERS:

The is.unsorted method for specific vector-like objects should adhere to the same underlying order used by the order, sort, and rank methods for the same objects.

See Also

• base::is.unsorted for the default is.unsorted method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• is.unsorted,GenomicRanges-method in the GenomicRanges package for an example of a specific is.unsorted method (defined for GenomicRanges objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

is.unsorted  # note the dispatch on the 'x' arg only
showMethods("is.unsorted")
selectMethod("is.unsorted", "ANY")  # the default method
lapply

Apply a function over a list-like or vector-like object

Description

lapply returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.
sapply is a user-friendly version and wrapper of lapply by default returning a vector, matrix or, if simplify="array", an array if appropriate, by applying simplify2array(). sapply(x, f, simplify=FALSE, USE.NAMES) is the same as lapply(x, f).

NOTE: This man page is for the lapply and sapply S4 generic functions defined in the BiocGenerics package. See ?base::lapply for the default methods (defined in the base package). Bioconductor packages can define specific methods for objects (typically list-like or vector-like) not supported by the default methods.

Usage

lapply(X, FUN, ...)
sapply(X, FUN, ..., simplify=TRUE, USE.NAMES=TRUE)

Arguments

X A list-like or vector-like object.
FUN, ..., simplify, USE.NAMES
See ?base::lapply for a description of these arguments.

Value

See ?base::lapply for the value returned by the default methods.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default methods. In particular, lapply and sapply(simplify=FALSE) should always return a list.

See Also

• base::lapply for the default lapply and sapply methods.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• lapply.List-method in the S4Vectors package for an example of a specific lapply method (defined for List objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

lapply # note the dispatch on the 'X' arg only
showMethods("lapply")
selectMethod("lapply", "ANY") # the default method

sapply # note the dispatch on the 'X' arg only
showMethods("sapply")
selectMethod("sapply", "ANY") # the default method
**lengths**

*Lengths of the list elements of a list-like object*

### Description

Get the length of each list element of a list-like object.

**NOTE:** This man page is for the `lengths` S4 generic function defined in the **BiocGenerics** package. See ?`base::lengths` for the default method (defined in the **base** package). Bioconductor packages can define specific methods for list-like objects not supported by the default method.

### Usage

```
lengths(x, use.names=TRUE)
```

### Arguments

- **x**
  - A list-like object. Can also be a vector-like object that is not list-like, in which case the result is trivial.
- **use.names**
  - See ?`base::lengths` for a description of this argument.

### Value

See ?`base::lengths` for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

### Note

**IMPORTANT:** The default method (base::lengths) is equivalent to `sapply(x, length)`. However, because the `lengths` method for Vector objects is currently defined as an alias for `S4Vectors::elementNROWS`, it's equivalent to `sapply(x, NROW)`, not to `sapply(x, length)`. This makes a difference if `x` has array-like list elements. See ?`base::NROW` for the difference between `length()` and `NROW()`. This difference is illustrated in the Examples section below.

This is a temporary situation that will be addressed in BioC 3.3.

### See Also

- `base::lengths` for the default lengths method.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `lengths,Vector-method` in the **S4Vectors** package for an example of a specific lengths method (defined for Vector objects).
- **BiocGenerics** for a summary of all the generics defined in the **BiocGenerics** package.
Examples

```r
lengths  # note the dispatch on the 'x' arg only
showMethods("lengths")
selectMethod("lengths", "ANY")  # the default method

library(S4Vectors)
showMethods("lengths")
selectMethod("lengths", "Vector")  # the "lengths" method for Vector # objects

## Difference between default method and method for Vector objects:
groups <- c("group1", "group2")

df <- data.frame(
    a=letters[1:10],
    i=101:110,
    group=rep(factor(groups, levels=groups), c(6, 4))
)
x1 <- split(df, df$group)
x2 <- split(DataFrame(df), df$group)
lengths(x1)  # dispatch on default method
lengths(x2)  # dispatch on method for Vector objects
```

---

**mad**

**Median Absolute Deviation**

Description

Compute the median absolute deviation for a vector, dispatching only on the first argument, `x`.

NOTE: This man page is for the `mad S4 generic function` defined in the **BiocGenerics** package. See `?stats::mad` for the default method (defined in the **stats** package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

```r
mad(x, center = median(x), constant = 1.4826,
    na.rm = FALSE, low = FALSE, high = FALSE)
```

Arguments

- `x`, `center`, `constant`, `na.rm`, `low`, `high`
  
  See `?stats::mad`.

Value

See `?stats::mad` for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.
mapply

See Also

- stats::mad for the default mad method.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

```
mad
showMethods("mad")
selectMethod("mad", "ANY")  # the default method
```

mapply

Apply a function to multiple list-like or vector-like arguments

Description

mapply is a multivariate version of sapply. mapply applies FUN to the first elements of each ... argument, the second elements, the third elements, and so on. Arguments are recycled if necessary.

NOTE: This man page is for the mapply S4 generic function defined in the BiocGenerics package. See ?base::mapply for the default method (defined in the base package). Bioconductor packages can define specific methods for objects (typically list-like or vector-like) not supported by the default methods.

Usage

```
mapply(FUN, ..., MoreArgs=NULL, SIMPLIFY=TRUE, USE.NAMES=TRUE)
```

Arguments

- FUN, MoreArgs, SIMPLIFY, USE.NAMES
- ... One or more list-like or vector-like objects of strictly positive length, or all of zero length.

Value

See ?base::mapply for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

See Also

- base::mapply for the default mapply method.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.
Examples

```r
mapply # note the dispatch on the '...' arg only
showMethods("mapply")
selectMethod("mapply", "ANY")  # the default method
```

match

---

Value matching

Description

match returns a vector of the positions of (first) matches of its first argument in its second.

NOTE: This man page is for the match S4 generic function defined in the BiocGenerics package. See ?base::match for the default method (defined in the base package). Bioconductor packages can define specific methods for objects (typically vector-like) not supported by the default method.

Usage

```r
match(x, table, nomatch=NA_integer_, incomparables=NULL, ...)
```

Arguments

- **x, table**: Vector-like objects (typically of the same class, but not necessarily).
- **nomatch, incomparables**: See ?base::match for a description of these arguments.
- **...**: Additional arguments, for use in specific methods.

Value

The same as the default method, that is, an integer vector of the same length as x giving the position in table of the first match if there is a match, otherwise nomatch.

See ?base::match for more details.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

Note

The default method (defined in the base package) doesn’t have the ... argument. We’ve added it to the generic function defined in the BiocGenerics package in order to allow specific methods to support additional arguments if needed.

See Also

- [base::match](#) for the default match method.
- [showMethods](#) for displaying a summary of the methods defined for a given generic function.
- [selectMethod](#) for getting the definition of a specific method.
- [match.Hits,Hits-method](#) in the S4Vectors package for an example of a specific match method (defined for Hits objects).
- [BiocGenerics](#) for a summary of all the generics defined in the BiocGenerics package.
normalize

Examples

```r
match # note the dispatch on the 'x' and 'table' args only
showMethods("match")
selectMethod("match", c("ANY", "ANY")) # the default method
```

normalize Normalize an object

Description

A generic function which normalizes an object containing microarray data or other data. Normalization is intended to remove from the intensity measures any systematic trends which arise from the microarray technology rather than from differences between the probes or between the target RNA samples hybridized to the arrays.

Usage

```r
normalize(object, ...)
```

Arguments

- `object` A data object, typically containing microarray data.
- `...` Additional arguments, for use in specific methods.

Value

An object containing the normalized data.

See Also

- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `normalize.AffyBatch-method` in the `affy` package and `normalize.FeatureSet-method` in the `oligo` package for examples of specific normalize methods (defined for AffyBatch and FeatureSet objects, respectively).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

```r
normalize
showMethods("normalize")
```

```r
library(affy)
showMethods("normalize")
selectMethod("normalize", "AffyBatch")
```
nrow  

The number of rows/columns of an array-like object

Description

Return the number of rows or columns present in an array-like object.

NOTE: This man page is for the nrow, ncol, NROW and NCOL S4 generic functions defined in the BiocGenerics package. See ?base::nrow for the default methods (defined in the base package). Bioconductor packages can define specific methods for objects (typically matrix- or array-like) not supported by the default methods.

Usage

nrow(x)
ncol(x)
NROW(x)
NCOL(x)

Arguments

x  
A matrix- or array-like object.

Value

A single integer or NULL.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default methods.

See Also

• base::nrow for the default nrow, ncol, NROW and NCOL methods.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• nrow.DataFrame-method in the S4Vectors package for an example of a specific nrow method (defined for DataFrame objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

nrow
showMethods("nrow")
selectMethod("nrow", "ANY")  # the default method

ncol
showMethods("ncol")
selectMethod("ncol", "ANY")  # the default method

NROW
showMethods("NROW")
selectMethod("NROW", "ANY")  # the default method
**Description**

`order` returns a permutation which rearranges its first argument into ascending or descending order, breaking ties by further arguments.

NOTE: This man page is for the `order` S4 generic function defined in the **BiocGenerics** package. See `?base::order` for the default method (defined in the **base** package). Bioconductor packages can define specific methods for objects (typically vector-like) not supported by the default method.

**Usage**

```r
order(..., na.last=TRUE, decreasing=FALSE, method=c("shell", "radix"))
```

**Arguments**

- `...`: One or more vector-like objects, all of the same length.
- `na.last`, `decreasing`, `method`
  
  See `?base::order` for a description of these arguments.

**Value**

The default method (see `?base::order`) returns an integer vector of length N where N is the common length of the input objects. This integer vector represents a permutation of N elements and can be used to rearrange the first argument in `...` into ascending or descending order (by subsetting it).

Specific methods defined in Bioconductor packages should also return an integer vector representing a permutation of N elements.

**Note**

TO DEVELOPERS:

Specific `order` methods should preferably be made "stable" for consistent behavior across platforms and consistency with `base::order()`. Note that C `qsort()` is not "stable" so `order` methods that use `qsort()` at the C-level need to ultimately break ties by position, which can easily be done by adding a little extra code at the end of the comparison function passed to `qsort()`.

`order(x, decreasing=TRUE)` is not always equivalent to `rev(order(x))`.

`order`, `sort`, and `rank` methods for specific vector-like objects should adhere to the same underlying order that should be conceptually defined as a binary relation on the set of all possible vector values. For completeness, this binary relation should also be incarnated by a `<=` method.
See Also

- `base::order` for the default `order` method.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `order.Ranges-method` in the `IRanges` package for an example of a specific `order` method (defined for `Ranges` objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

```r
order
showMethods("order")
selectMethod("order", "ANY") # the default method
```

---

**organism_species**  
Organism and species accessors

Description

Get or set the organism and/or species of an object.

Usage

```r
organism(object)
organism(object) <- value

species(object)
species(object) <- value
```

Arguments

- `object`  
  An object to get or set the organism or species of.

- `value`  
  The organism or species to set on `object`.

Value

- `organism` should return the *scientific name* (i.e. genus and species, or genus and species and subspecies) of the organism. Preferably in the format "Genus species" (e.g. "Homo sapiens") or "Genus species subspecies" (e.g. "Homo sapiens neanderthalensis").

- `species` should of course return the species of the organism. Unfortunately there is a long history of misuse of this accessor in Bioconductor so its usage is now discouraged (starting with BioC 3.1).

Note

TO DEVELOPERS:

- `species` has been historically misused in many places in Bioconductor and is redundant with `organism`. So implementing the `species` accessor is now discouraged (starting with BioC 3.1).
- The `organism` accessor (returning the *scientific name*) should be implemented instead.
paste

Concatenate strings

Description

paste concatenates vectors of strings or vector-like objects containing strings.

NOTE: This man page is for the \texttt{paste} \textit{S4 generic function} defined in the \texttt{BiocGenerics} package. See \texttt{?base::paste} for the default method (defined in the \texttt{base} package). Bioconductor packages can define specific methods for objects (typically vector-like objects containing strings) not supported by the default method.

Usage

\begin{verbatim}
paste(..., sep=" ", collapse=NULL)
\end{verbatim}
plotMA

Arguments

... One or more vector-like objects containing strings.
sep, collapse See ?base::paste for a description of these arguments.

Value

See ?base::paste for the value returned by the default method.

Specific methods defined in Bioconductor packages will typically return an object of the same class as the input objects.

See Also

• base::paste for the default paste method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• paste.Rle-method in the S4Vectors package for an example of a specific paste method (defined for Rle objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

plotMA

showMethods("paste")
selectMethod("paste", "ANY") # the default method

plotMA                      MA-plot: plot differences versus averages for high-throughput data

Description

A generic function which produces an MA-plot for an object containing microarray, RNA-Seq or other data.

Usage

plotMA(object, ...)

Arguments

object A data object, typically containing count values from an RNA-Seq experiment or microarray intensity values.

... Additional arguments, for use in specific methods.

Value

Undefined. The function exists for its side effect, producing a plot.
See Also

- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `plotMA` in the `limma` package for a function with the same name that is not dispatched through this generic function.
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

```r
showMethods("plotMA")
suppressWarnings(
  if(require("DESeq2"))
    example("plotMA", package="DESeq2", local=TRUE)
)
```
**rank**

Ranks the values in a vector-like object

**Description**

Returns the ranks of the values in a vector-like object. Ties (i.e., equal values) and missing values can be handled in several ways.

NOTE: This man page is for the rank S4 generic function defined in the BiocGenerics package. See ?base::rank for the default method (defined in the base package). Bioconductor packages can define specific methods for objects not supported by the default method.

**Usage**

```r
rank(x, na.last=TRUE,
     ties.method=c("average", "first", "last", "random", "max", "min"))
```

**Arguments**

- `x`: A vector-like object.
- `na.last`, `ties.method`: See ?base::rank for a description of these arguments.

**Value**

See ?base::rank for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

**Note**

TO DEVELOPERS:

See note in ?BiocGenerics::order about "stable" order.

order, sort, and rank methods for specific vector-like objects should adhere to the same underlying order that should be conceptually defined as a binary relation on the set of all possible vector values. For completeness, this binary relation should also be incarnated by a <= method.

**See Also**

- base::rank for the default rank method.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- rank.Vector-method in the S4Vectors package for an example of a specific rank method (defined for Vector objects).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

**Examples**

```r
rank  # note the dispatch on the 'x' arg only
showMethods("rank")
selectMethod("rank", "ANY")  # the default method
```
Description

relist is a generic function with a few methods in order to allow easy inversion of unlist(x).

NOTE: This man page is for the relist S4 generic function defined in the BiocGenerics package. See ?utils::relist for the default method (defined in the utils package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

relist(flesh, skeleton)

Arguments

flesh A vector-like object.
skeleton A list-like object. Only the “shape” (i.e. the lengths of the individual list elements) of skeleton matters. Its exact content is ignored.

Value

A list-like object with the same "shape" as skeleton and that would give flesh back if unlist(ed).

See Also

• utils::relist for the default relist method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• relist.ANY.List-method in the IRanges package for an example of a specific relist method (defined for when skeleton is a List object).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

relist
showMethods("relist")
selectMethod("relist", c("ANY", "ANY")) # the default method
rep.int rep.int replicates the elements in x.

NOTE: This man page is for the rep.int S4 generic function defined in the BiocGenerics package. See ?base::rep.int for the default method (defined in the base package). Bioconductor packages can define specific methods for objects (typically vector-like) not supported by the default method.

Usage

rep.int(x, times)

Arguments

x The object to replicate (typically vector-like).
times See ?base::rep.int for a description of this argument.

Value

See ?base::rep.int for the value returned by the default method.

Specific methods defined in Bioconductor packages will typically return an object of the same class as the input object.

See Also

• base::rep.int for the default rep.int, intersect, and setdiff methods.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• rep.int.Rle-method in the S4Vectors package for an example of a specific rep.int method (defined for Rle objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

rep.int
showMethods("rep.int")
selectMethod("rep.int", "ANY") # the default method
residuals

Extract model residuals

Description

residuals is a generic function which extracts model residuals from objects returned by modeling functions.

NOTE: This man page is for the residuals S4 generic function defined in the BiocGenerics package. See ?stats::residuals for the default method (defined in the stats package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

residuals(object, ...)

Arguments

object, ... See ?stats::residuals.

Value

Residuals extracted from the object object.

See Also

• stats::residuals for the default residuals method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• residuals.PLMset-method in the affyPLM package for an example of a specific residuals method (defined for PLMset objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

residuals
showMethods("residuals")
selectMethod("residuals", "ANY") # the default method

row+colnames

Row and column names

Description

Get or set the row or column names of a matrix-like object.

NOTE: This man page is for the rownames, `rownames<~`, colnames, and `colnames<~` S4 generic functions defined in the BiocGenerics package. See ?base::rownames for the default methods (defined in the base package). Bioconductor packages can define specific methods for objects (typically matrix-like) not supported by the default methods.
Usage

rownames(x, do.NULL=TRUE, prefix="row")
rownames(x) <- value

colnames(x, do.NULL=TRUE, prefix="col")
colnames(x) <- value

Arguments

x                A matrix-like object.
do.NULL, prefix  See ?base::rownames for a description of these arguments.
value            Either NULL or a character vector equal of length equal to the appropriate dimension.

Value

The getters will return NULL or a character vector of length \texttt{nrow}(x) for rownames and length \texttt{ncol}(x) for colnames(x).

See \texttt{?base::rownames} for more information about the default methods, including how the setters are expected to behave.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default methods.

See Also

• \texttt{base::rownames} for the default \texttt{rownames}, \texttt{rownames<-}, \texttt{colnames}, and \texttt{colnames<-} methods.
• \texttt{showMethods} for displaying a summary of the methods defined for a given generic function.
• \texttt{selectMethod} for getting the definition of a specific method.
• \texttt{rownames,DataFrame-method} in the \texttt{S4Vectors} package for an example of a specific \texttt{rownames} method (defined for \texttt{DataFrame} objects).
• \texttt{BiocGenerics} for a summary of all the generics defined in the \texttt{BiocGenerics} package.

Examples

```r
## rownames getter:
rownames # note the dispatch on the 'x' arg only
showMethods("rownames")
selectMethod("rownames", "ANY") # the default method

## rownames setter:
rownames<- # note the dispatch on the 'x' arg only
showMethods("rownames<-")
selectMethod("rownames<-", "ANY") # the default method

## colnames getter:
colnames # note the dispatch on the 'x' arg only
showMethods("colnames")
selectMethod("colnames", "ANY") # the default method

## colnames setter:
```
Some old-style (aka S3) classes are turned into formally defined (aka S4) classes by the BiocGenerics package. This allows S4 methods defined in Bioconductor packages to use them in their signatures.

S3 classes currently turned into S4 classes:

* connection class and subclasses: connection, file, url, gzfile, bzfile, unz, pipe, fifo, sockconn, terminal, textConnection, gzcon. Additionally the characterORconnection S4 class is defined as the union of classes character and connection.
* others: AsIs, dist

See Also

setOldClass and setClassUnion in the methods package.

score

Score accessor

Description

Get or set the score value contained in an object.

Usage

score(x, ...)
score(x, ...) <- value

Arguments

x An object to get or set the score value of.
... Additional arguments, for use in specific methods.
value The score value to set on x.
See Also

- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `score.GenomicRanges-method` in the `GenomicRanges` package for an example of a specific score method (defined for `GenomicRanges` objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

```r
score
showMethods("score")

`score<-`
showMethods("score<-")

library(GenomicRanges)
showMethods("score")
selectMethod("score", "GenomicRanges")
showMethods("score<-")
selectMethod("score<-", "GenomicRanges")
```

Description

Performs set union, intersection and (asymmetric!) difference on two vector-like objects.

NOTE: This man page is for the `union`, `intersect` and `setdiff` S4 generic functions defined in the `BiocGenerics` package. See `?base::union` for the default methods (defined in the `base` package). Bioconductor packages can define specific methods for objects (typically vector-like) not supported by the default methods.

Usage

```r
union(x, y, ...)
intersect(x, y, ...)
setdiff(x, y, ...)
```

Arguments

- `x, y`  Vector-like objects (typically of the same class, but not necessarily).
- `...`  Additional arguments, for use in specific methods.

Value

See `?base::union` for the value returned by the default methods.

Specific methods defined in Bioconductor packages will typically return an object of the same class as the input objects.
Note

The default methods (defined in the base package) only take 2 arguments. We’ve added the ... argument to the generic functions defined in the BioGenerics package so they can be called with an arbitrary number of effective arguments. For union or intersect, this typically allows Bioconductor packages to define methods that compute the union or intersection of more than 2 objects. However, for setdiff, which is conceptually a binary operation, this typically allows methods to add extra arguments for controlling/altering the behavior of the operation. Like for example the ignore.strand argument supported by the setdiff method for GRanges objects (defined in the GenomicRanges package). (Note that the union and intersect methods for those objects also support the ignore.strand argument.)

See Also

- base::union for the default union, intersect, and setdiff methods.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- union,GRanges,GRanges-method in the GenomicRanges package for an example of a specific union method (defined for GRanges objects).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

```r
union
showMethods("union")
selectMethod("union", c("ANY", "ANY")) # the default method

intersect
showMethods("intersect")
selectMethod("intersect", c("ANY", "ANY")) # the default method

setdiff
showMethods("setdiff")
selectMethod("setdiff", c("ANY", "ANY")) # the default method
```

sort

**Sorting a vector-like object**

Description

Sort a vector-like object into ascending or descending order.

NOTE: This man page is for the sort S4 generic function defined in the BiocGenerics package. See ?base::sort for the default method (defined in the base package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

```r
sort(x, decreasing=FALSE, ...)
```
Arguments

x  A vector-like object.
  decreasing, ...
  See ?base::sort for a description of these arguments.

Value

See ?base::sort for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

Note

TO DEVELOPERS:

See note in ?BiocGenerics::order about "stable" order.

order, sort, and rank methods for specific vector-like objects should adhere to the same underlying order that should be conceptually defined as a binary relation on the set of all possible vector values. For completeness, this binary relation should also be incarnated by a <= method.

See Also

• base::sort for the default sort method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• sort,Vector-method in the S4Vectors package for an example of a specific sort method (defined for Vector objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

sort # note the dispatch on the 'x' arg only
showMethods("sort")
selectMethod("sort", "ANY") # the default method

---------------------------------------------

start                      Start, end, width generic getters and setters

---------------------------------------------

Description

Get or set the start, end, or width of an object.

NOTE: This man page is for the start, `start<-`, end, `end<-`, width, and `width<-` S4 generic functions defined in the BiocGenerics package. See ?stats::start for the start and end S3 generics defined in the stats package.
start

Usage

\begin{verbatim}
start(x, ...) <- value
\end{verbatim}

\begin{verbatim}
end(x, ...) <- value
\end{verbatim}

\begin{verbatim}
width(x) <- value
\end{verbatim}

Arguments

\begin{verbatim}
x
\end{verbatim}

An object containing start, end, and width values.

\begin{verbatim}
...
\end{verbatim}

Additional arguments, for use in specific methods.

\begin{verbatim}
value
\end{verbatim}

The start, end, or width values to set on \texttt{x}.

Value

See specific methods defined in Bioconductor packages.

See Also

- \texttt{stats::start} in the \texttt{stats} package for the \texttt{start} and \texttt{end} S3 generics.
- \texttt{showMethods} for displaying a summary of the methods defined for a given generic function.
- \texttt{selectMethod} for getting the definition of a specific method.
- \texttt{start,IRanges-method} in the \texttt{IRanges} package for examples of specific \texttt{start}, \texttt{end}, and \texttt{width} methods (defined for \texttt{IRanges} objects).
- \texttt{BiocGenerics} for a summary of all the generics defined in the \texttt{BiocGenerics} package.

Examples

```r
## start getter:
start
showMethods("start")

library(IRanges)
showMethods("start")
selectMethod("start", "IRanges") # start getter for IRanges objects

## start setter:
\`
start<-
\`
showMethods("start<-")
selectMethod("start<-", "IRanges") # start setter for IRanges objects

## end getter:
end
showMethods("end")
selectMethod("end", "IRanges") # end getter for IRanges objects

## end setter:
\`
end<-
\`
showMethods("end<-")
selectMethod("end<-", "IRanges") # end setter for IRanges objects
```
strand

Accessing strand information

Description

Get or set the strand information contained in an object.

Usage

strand(x, ...)
strand(x, ...) <- value

unstrand(x)

invertStrand(x)

## S4 method for signature 'ANY'
invertStrand(x)

Arguments

x An object containing strand information.
...
value The strand information to set on x.

Details

All the strand methods defined in the GenomicRanges package use the same set of 3 values (called the "standard strand levels") to specify the strand of a genomic location: +, -, and *. * is used when the exact strand of the location is unknown, or irrelevant, or when the "feature" at that location belongs to both strands.

Note that unstrand is not a generic function, just a convenience wrapper to the generic strand setter (strand<-) that does:

strand(x) <- "*

x

The default method for invertStrand does:

strand(x) <- invertStrand(strand(x))

x
If x is a vector-like object, strand(x) will typically return a vector-like object parallel to x, that is, an object of the same length as x where the i-th element describes the strand of the i-th element in x.

unstrand(x) and invertStrand(x) return a copy of x with the strand set to "*" for unstrand or inverted for invertStrand (i.e. "+" and "-" switched, and "*" untouched).

See Also

- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- strand.GRanges-method in the GenomicRanges package for an example of a specific strand method (defined for GRanges objects).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

```
strand
showMethods("strand")

\`strand<-`  
showMethods("strand<-")

unstrand

invertStrand  
showMethods("invertStrand")
selectMethod("invertStrand", "ANY")  # the default method

library(GenomicRanges)

showMethods("strand")
selectMethod("strand", "missing")
strand()
showMethods("strand<-")
```

Description

Return subsets of vector-like, matrix-like or data-frame-like objects which meet conditions.

NOTE: This man page is for the subset S4 generic function defined in the BiocGenerics package. See ?base::subset for the subset S3 generic defined in the base package.

Usage

```
subset(x, ...)
```
Arguments

- `x`: A vector-like, matrix-like or data-frame-like object to be subsetted.
- `...`: Additional arguments (e.g. `subset`, `select`, `drop`), for use in specific methods. See `?base::subset` for more information.

Value

An object similar to `x` containing just the selected elements (for a vector-like object), or the selected rows and columns (for a matrix-like or data-frame-like object).

See Also

- `base::subset` in the `base` package for the `subset` S3 generic.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `subset,DataTable-method` in the `S4Vectors` package for an example of a specific `subset` method (defined for `DataTable` objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

```r
subset
showMethods("subset")
selectMethod("subset", "ANY") # the default method

library(S4Vectors)
showMethods("subset")
selectMethod("subset", "DataTable") # the "subset" method for
# DataTable objects
```
Value

See ?base::table for the value returned by the default method.
Specific methods defined in Bioconductor packages should also return the type of object returned by the default method.

See Also

- base::table for the default table method.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- table.Rle-method in the S4Vectors package for an example of a specific table method (defined for Rle objects).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

table
showMethods("table")
selectMethod("table", "ANY")  # the default method

description

tapply applies a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

NOTE: This man page is for the tapply S4 generic function defined in the BiocGenerics package. See ?base::tapply for the default method (defined in the base package). Bioconductor packages can define specific methods for objects (typically list-like or vector-like) not supported by the default method.

Usage

tapply(X, INDEX, FUN=NULL, ..., simplify=TRUE)

Arguments

X  The default method expects an atomic object, typically a vector. See ?base::tapply for the details.
Specific methods can support other objects (typically list-like or vector-like). Please refer to the documentation of a particular method for the details.

INDEX  The default method expects a list of one or more factors, each of same length as X. See ?base::tapply for the details.
Specific methods can support other objects (typically list-like). Please refer to the documentation of a particular method for the details.

FUN, ..., simplify  See ?base::tapply for a description of these arguments.
Value

See \?base::tapply for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

See Also

- base::tapply for the default \texttt{tapply} method.
- \texttt{showMethods} for displaying a summary of the methods defined for a given generic function.
- \texttt{selectMethod} for getting the definition of a specific method.
- \texttt{tapply,Vector,ANY-method} in the \texttt{IRanges} package for an example of a specific \texttt{tapply} method (defined for \texttt{Vector} objects).
- \texttt{BiocGenerics} for a summary of all the generics defined in the \texttt{BiocGenerics} package.

Examples

tapply  # note the dispatch on the 'X' and 'INDEX' args only
showMethods("tapply")
selectMethod("tapply", c("ANY", "ANY"))  # the default method

---

\textbf{testPackage} \hspace{1cm} \textit{Run RUnit package unit tests}

Description

testPackage helps developers implement unit tests using the \texttt{RUnit} testing conventions.

Usage

testPackage(pkname=NULL, subdir="unitTests", pattern="^test_.*\.R$", path=getwd())

Arguments

\begin{itemize}
\item \textbf{pkname} \hspace{1cm} The name of the package whose installed unit tests are to be run. A missing or NULL value implies that the testPackage command will look for tests within the package source directory indicated by \texttt{path}.
\item \textbf{subdir} \hspace{1cm} A character(1) vector providing the subdirectory in which unit tests are located. The directory is searched first in the (installed or source) package root, or in a subdirectory \texttt{inst/} below the root.
\item \textbf{pattern} \hspace{1cm} A character(1) regular expression describing the file names to be evaluated; typically used to restrict tests to a subset of all test files.
\item \textbf{path} \hspace{1cm} A character(1) directory path indicating, when \texttt{pkname} is missing or NULL, where unit tests will be searched. \texttt{path} can be any location at or below the package root.
\end{itemize}
Details

This function is not exported from the package namespace, and must be invoked using triple colons, `BiocGenerics:::testPackage()`. It is provided primarily for the convenience of developers.

When invoked with missing or NULL `pkgname` argument, the function assumes that it has been invoked from within the package source tree (or that the source tree is located above `path`), and finds unit tests in `subdir="unitTests"` in either the base or `inst/` directories at the root of the package source tree. This mode is useful when developing unit tests, since the package does not have to be re-installed to run an updated test.

When invoked with `pkgname` set to the name of an installed package, unit tests are searched for in the installed package directory.

Value

The function returns the result of `RUnit::runTestSuite` invoked on the unit tests specified in the function call.

See Also

http://bioconductor.org/developers/how-to/unitTesting-guidelines/

Examples

```r
## Run unit tests found in the library location where
## BiocGenerics is installed
BiocGenerics:::testPackage("BiocGenerics")
## Not run: ## Run unit tests for the package whose source tree implied
## by getwd()
BiocGenerics:::testPackage()

## End(Not run)
```

<table>
<thead>
<tr>
<th>unique</th>
<th>Extract unique elements</th>
</tr>
</thead>
</table>

Description

unique returns an object of the same class as `x` (typically a vector-like, data-frame-like, or array-like object) but with duplicate elements/rows removed.

NOTE: This man page is for the `unique` S4 generic function defined in the `BiocGenerics` package. See `?base::unique` for the default method (defined in the `base` package). Bioconductor packages can define specific methods for objects (typically vector-like or data-frame-like) not supported by the default method.

Usage

```r
unique(x, incomparables=FALSE, ...)
```

Arguments

- `x` A vector-like, data-frame-like, or array-like object.
- `incomparables`, `...` See `?base::unique` for a description of these arguments.
Value

See ?base::unique for the value returned by the default method.

Specific methods defined in Bioconductor packages will typically return an object of the same class as the input object.

unique should always behave consistently with BiocGenerics::duplicated.

See Also

• base::unique for the default unique method.
• BiocGenerics::duplicated for determining duplicate elements.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• unique.Rle-method in the S4Vectors package for an example of a specific unique method (defined for Rle objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

unique
showMethods("unique")
selectMethod("unique", "ANY") # the default method

unlist

Flatten list-like objects

Description

Given a list-like object x, unlist produces a vector-like object obtained by concatenating (conceptually thru c) all the top-level elements in x (each of them being expected to be a vector-like object, typically).

NOTE: This man page is for the unlist S4 generic function defined in the BiocGenerics package. See ?base::unlist for the default method (defined in the base package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

unlist(x, recursive=TRUE, use.names=TRUE)

Arguments

x A list-like object.
recursive, use.names

See ?base::unlist for a description of these arguments.

Value

See ?base::unlist for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.
See Also

- `base::unlist` for the default unlist method.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `unlist,List-method` in the `S4Vectors` package for an example of a specific unlist method (defined for `List` objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

unlist  # note the dispatch on the 'x' arg only
showMethods("unlist")
selectMethod("unlist", "ANY")  # the default method

unsplit  

Unsplit a list-like object

Description

Given a list-like object `value` and grouping `f`, `unsplit` produces a vector-like object `x` by conceptually reversing the split operation `value <- split(x, f)`.

NOTE: This man page is for the `unsplit` `S4 generic function` defined in the `BiocGenerics` package. See `?base::unsplit` for the default method (defined in the `base` package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

unsplit(value, f, drop=FALSE)

Arguments

value  
A list-like object.

f  
A factor or other grouping object that corresponds to the `f` symbol in `value <- split(x, f)`.

drop  
See `?base::unsplit` for a description of this argument.

Value

See `?base::unsplit` for the value returned by the default method.

Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

See Also

- `base::unsplit` for the default unsplit method.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `unsplit,List-method` in the `IRanges` package for an example of a specific unsplit method (defined for `List` objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.
Examples

unsplit # note the dispatch on the 'value' and 'f' args only
showMethods("unsplit")
selectMethod("unsplit", "ANY")  # the default method

updateObject

Update an object to its current class definition

Description

updateObject is a generic function that returns an instance of object updated to its current class definition.

Usage

updateObject(object, ..., verbose=FALSE)

## Related utilities:
updateObjectFromSlots(object, objclass=class(object), ..., verbose=FALSE)
getObjectSlots(object)

Arguments

object Object to be updated for updateObject and updateObjectFromSlots.
Object for slot information to be extracted from for getObjectSlots.
...
Additional arguments, for use in specific updateObject methods.
verbose TRUE or FALSE, indicating whether information about the update should be reported. Use message to report this information.
objclass Optional character string naming the class of the object to be created.

Details

Updating objects is primarily useful when an object has been serialized (e.g., stored to disk) for some time (e.g., months), and the class definition has in the mean time changed. Because of the changed class definition, the serialized instance is no longer valid.

updateObject requires that the class of the returned object be the same as the class of the argument object, and that the object is valid (see validObject). By default, updateObject has the following behaviors:

updateObject(ANY, ..., verbose=FALSE) By default, updateObject uses heuristic methods to determine whether the object should be the ‘new’ S4 type (introduced in R 2.4.0), but is not. If the heuristics indicate an update is required, the updateObjectFromSlots function tries to update the object. The default method returns the original S4 object or the successfully updated object, or issues an error if an update is required but not possible. The optional named argument verbose causes a message to be printed describing the action. Arguments ... are passed to updateObjectFromSlots.

updateObject(list, ..., verbose=FALSE) Visit each element in list, applying updateObject(list[[elt]], ...) to each element.
updateObject(environment, ..., verbose=FALSE) Visit each element in environment, applying updateObject(environment[[elt]], ..., verbose=verbose) to each element.
**updateObject**

updateObject(formula, ..., verbose=FALSE) Do nothing; the environment of the formula may be too general (e.g., R_GlobalEnv) to attempt an update.

updateObject(envRefClass, ..., verbose=FALSE) Attempt to update objects from fields using a strategy like updateObjectFromSlots Method 1.

updateObjectFromSlots(object, objclass=class(object), ..., verbose=FALSE) is a utility function that identifies the intersection of slots defined in the object instance and objclass definition. Under Method 1, the corresponding elements in object are then updated (with updateObject(elt, ..., verbose=verbose)) and used as arguments to a call to new(class, ...), with ... replaced by slots from the original object. If this fails, then Method 2 tries new(class) and assigns slots of object to the newly created instance.

getObjectSlots(object) extracts the slot names and contents from object. This is useful when object was created by a class definition that is no longer current, and hence the contents of object cannot be determined by accessing known slots.

**Value**

updateObject returns a valid instance of object.

updateObjectFromSlots returns an instance of class objclass.

getObjectSlots returns a list of named elements, with each element corresponding to a slot in object.

**See Also**

- updateObjectTo in the Biobase package for updating an object to the class definition of a template (might be useful for updating a virtual superclass).
- validObject for testing the validity of an object.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

**Examples**

```r
updateObject
showMethods("updateObject")
selectMethod("updateObject", "ANY") # the default method

library(Biobase)
## update object, same class
data(sample.ExpressionSet)
obj <- updateObject(sample.ExpressionSet)

setClass("UpdtA", representation(x="numeric"), contains="data.frame")
setMethod("updateObject", "UpdtA", 
  function(object, ..., verbose=FALSE) 
  { 
    if (verbose
      message("updateObject object = 'A'")
    object <- callNextMethod()
    object$x <- -object$x
    object
    }
  }
)
```
weights

Description

weights is a generic function which extracts fitting weights from objects returned by modeling functions.

NOTE: This man page is for the weights S4 generic function defined in the Bioconductor package. See ?stats::weights for the default method (defined in the stats package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

weights(object, ...)

Arguments

object, ... See ?stats::weights.

Value

Weights extracted from the object object.
See ?stats::weights for the value returned by the default method.
Specific methods defined in Bioconductor packages should behave as consistently as possible with the default method.

See Also

• stats::weights for the default weights method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• weights.PLMSet-method in the affyPLM package for an example of a specific weights method (defined for PLMSet objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

weights
showMethods("weights")
selectMethod("weights", "ANY") # the default method
which

Subscript generators

Description

These functions all return a vector of subscripts into their input.

NOTE: This man page is for the which, which.max and which.min S4 generic functions defined in the BiocGenerics package. See ?base::which and ?base::which.min for the default methods (defined in the base package). Bioconductor packages can define specific methods for objects (typically vector-like) not supported by the default methods.

Usage

which(x, arr.ind = FALSE, useNames = TRUE, ...)  
which.max(x, ...)  
which.min(x, ...)

Arguments

x
Vector-like object, logical for which, numeric for the others.

arr.ind, useNames
See ?base::which for a description of these arguments.

Additional arguments, for use in specific methods.

Value

See ?base::which and ?base::which.min for the value returned by the default methods.

Specific methods defined in Bioconductor packages will typically return an object of the same class as the input objects.

Note

The default methods (defined in the base package) only take a fixed set of arguments. We’ve added the ... argument to the generic functions defined in the BiocGenerics package so they can be called with an arbitrary number of effective arguments. This typically allows methods to add extra arguments for controlling/altering the behavior of the operation. Like for example the global argument supported by the which.max method for NumericList objects (defined in the IRanges package).

See Also

• base::which for the default which, base::which.min for the others.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• which.max.NumericList-method in the IRanges package for an example of a specific which.max method (defined for NumericList objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.
Examples

which
showMethods("which")
selectMethod("which", c("ANY", "ANY"))  # the default method

which.max
showMethods("which.max")
selectMethod("which.max", c("ANY", "ANY"))  # the default method

which.min
showMethods("which.min")
selectMethod("which.min", c("ANY", "ANY"))  # the default method

xtabs

Cross tabulation

Description

xtabs creates a contingency table (optionally a sparse matrix) from cross-classifying factors, usually contained in a data-frame-like object, using a formula interface.

NOTE: This man page is for the xtabs S4 generic function defined in the BiocGenerics package. See ?stats::xtabs for the default method (defined in the stats package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

xtabs(formula=~., data=parent.frame(), subset, sparse=FALSE,
       na.action, exclude=c(NA, NaN), drop.unused.levels=FALSE)

Arguments

formula, subset, sparse, na.action, exclude, drop.unused.levels
See ?stats::xtabs for a description of these arguments.

data  A data-frame-like object.

Value

See ?stats::xtabs for the value returned by the default method.

Specific methods defined in Bioconductor packages should also return the type of object returned by the default method.

See Also

• stats::xtabs for the default xtabs method.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• xtabs.DataTable-method in the S4Vectors package for an example of a specific xtabs method (defined for DataTable objects).
• BiocGenerics for a summary of all the generics defined in the BiocGenerics package.
Examples

xtabs  # note the dispatch on the 'data' arg only
showMethods("xtabs")
selectMethod("xtabs", "ANY")  # the default method
Index

*Topic **classes**
  S3-classes-as-S4-classes, 47
*Topic **manip**
  dge, 17
*Topic **methods**
  annotation, 5
  append, 6
  as.data.frame, 7
  as.vector, 8
  boxplot, 9
  cbind, 10
  clusterApply, 11
  combine, 13
  dbconn, 15
  density, 16
  do.call, 17
  duplicated, 18
  eval, 20
  Extremes, 21
  fileName, 23
  funprog, 23
  get, 25
  grep, 26
  image, 27
  IQR, 28
  is.unsorted, 28
  lapply, 30
  lengths, 31
  mad, 32
  mapply, 33
  match, 34
  normalize, 35
  nrow, 36
  order, 37
  organism_species, 38
  paste, 39
  plotMA, 40
  plotPCA, 41
  rank, 42
  relist, 43
  rep, 44
  residuals, 45
  row+colnames, 45
  score, 47
  sets, 48
  sort, 49
  start, 50
  strand, 52
  subset, 53
  table, 54
  tapply, 55
  testPackage, 56
  unique, 57
  unlist, 58
  unsplit, 59
  updateObject, 60
  weights, 62
  which, 63
  xtabs, 64
*Topic **package**
  BiocGenerics-package, 3
  <=, 37, 42, 50
  AffyBatch, 35
  AnnotatedDataFrame, 14
  annotation, 4, 5
  annotation,eSet-method, 5
  annotation<- (annotation), 5
  AnnotationDb, 39
  anyDuplicated, 3
  anyDuplicated (duplicated), 18
  append, 3, 6, 6
  append,Vector,Vector-method, 6
  as.data.frame, 3, 7, 7
  as.data.frame,DataFrame-method, 7
  as.data.frame,Ranges-method, 7
  as.vector, 3, 8, 8
  as.vector,AtomicList-method, 8
  as.vector,Rle-method, 8
  AsIs, 47
  AsIs-class(S3-classes-as-S4-classes), 47
  AssayData, 14
  AtomicList, 8
  Bimap, 25

BiocGenerics (BiocGenerics-package), 3
BiocGenerics-package, 3
boxplot, 4, 9, 9
boxplot,FEATURESet-method, 9
bzfile-class
(S3-classes-as-S4-classes), 47
c, 58
cbind, 3, 4, 10, 10
cbind,DataFrame-method, 10
characterORconnection-class
(S3-classes-as-S4-classes), 47
chrmlnLocation, 39
clusterApply, 4, 11, 11, 12
clusterApplyLB, 4
clusterApplyLB(clusterApply), 11
clusterCall, 4
clusterCall(clusterApply), 11
clusterEvalQ, 4
clusterEvalQ(clusterApply), 11
clusterExport, 4
clusterExport(clusterApply), 11
clusterMap, 4
clusterMap(clusterApply), 11
clusterSplit, 4
clusterSplit(clusterApply), 11
colnames, 3
colnames(row+colnames), 45
colnames<- (row+colnames), 45
combine, 4, 13
combine,AnnotatedDataFrame,AnnotatedDataFrame-method, 14
combine,ANY,missing-method (combine), 13
combine,AssayData,AssayData-method, 14
combine,data.frame,data.frame-method (combine), 13
combine,eSet,eSet-method, 14
combine,matrix,matrix-method (combine), 13
combine,MIAME,MIAME-method, 14
conditions (dge), 17
conditions<- (dge), 17
connection, 47
connection-class
(S3-classes-as-S4-classes), 47
counts (dge), 17
counts<- (dge), 17

DataFrame, 7, 10, 36, 46
DataTable, 54, 64
dbconn, 4, 15, 15
dbconn,AnnotationDb-method, 15
dbfile, 4
dbfile (dbconn), 15
density, 4, 16, 16
density,flowClust-method, 16
design (dge), 17
design<- (dge), 17
dge, 17
dim, 4
dispTable (dge), 17
dispTable<- (dge), 17
dist, 47
dist-class (S3-classes-as-S4-classes), 47
do.call, 3, 17, 17, 18
duplicated, 3, 18, 18, 19, 58
duplicated,Rle-method, 19
elementNROWS, 31
end, 3
end (start), 50
de<-(start), 50
eSet, 5, 14
estimateDispersions (dge), 17
estimateSizeFactors (dge), 17
eval, 3, 20, 20, 21
eval,expression,Vector-method, 20
evalq, 21, 21
expression, 20
Extremes, 21
factor, 14
FeatureSet, 9, 27, 35
fifo-class (S3-classes-as-S4-classes), 47
file-class (S3-classes-as-S4-classes), 47
fileName, 4, 23
fileName,MSmap-method, 23
Filter, 3
Filter (funprog), 23
Find, 3
Find (funprog), 23
flowClust, 16
funprog, 23
GenomicRanges, 29, 48
gc, 3, 25, 25
gc,ANY, Bimap,missing-method, 25
goB,objectSlots (updateObject), 60
GRanges, 49, 53
grep, 3, 26, 26
INDEX

grepl, 3
grepl (grep), 26
groupGeneric, 4
gzcon-class (S3-classes-as-S4-classes), 47
gzfile-class
(S3-classes-as-S4-classes), 47

Hits, 34

image, 4, 27
image, FeatureSet-method, 27
InternalMethods, 4
intersect, 3
intersect (sets), 48
invertStrand, 4
invertStrand (strand), 52
invertStrand, ANY-method (strand), 52
IQR, 28, 28
IRanges, 51
is.unsorted, 3, 28, 28, 29
is.unsorted, GenomicRanges-method, 29

lapply, 3, 30, 30
lapply, List-method, 30
length, 4
lengths, 3, 31, 31
lengths, Vector-method, 31
List, 24, 30, 43, 59

mad, 32, 32, 33
Map, 3
Map (funprog), 23
mapply, 3, 33, 33
match, 3, 34, 34
match, Hits, Hits-method, 34
Math, 4
merge, 14
message, 60
mget, 3
mget (get), 25
MIAME, 14
MSmap, 23

NCOL, 3
NCOL (nrow), 36
ncol, 3, 46
ncol (nrow), 36
normalize, 4, 35
normalize, AffyBatch-method, 35
normalize, FeatureSet-method, 35
NROW, 3, 31
NROW (nrow), 36

nrow, 3, 36, 36, 46
nrow, DataFrame-method, 36
NumericList, 63

Ops, 4
order, 3, 29, 37, 37, 38, 42, 50
order, Ranges-method, 38
organism, 4
organism (organism_species), 38
organism, character-method, 39
organism, chromLocation-species, 39
organism<- (organism_species), 38
organism_species, 38

parApply, 4
parApply (clusterApply), 11
parCapply, 4
parCapply (clusterApply), 11
parLapply, 4
parLapply (clusterApply), 11
parLapplyLB, 4
parLapplyLB (clusterApply), 11
parRapply, 4
parRapply (clusterApply), 11
parSapply, 4
parSapply (clusterApply), 11
parSapplyLB, 4
parSapplyLB (clusterApply), 11
paste, 3, 39, 39, 40
paste, Rle-method, 40
pipe-class (S3-classes-as-S4-classes), 47

PLMset, 45, 62
plotDispEsts (dge), 17
plotMA, 4, 40, 41
plotMA, ANY-method (plotMA), 40
plotPCA, 4, 41, 41
plotPCA, ANY-method (plotPCA), 41
pmax, 3, 21, 22
pmax (Extremes), 21
pmax, Rle-method, 22
pmax.int, 3
pmin, 3
pmin (Extremes), 21
pmin.int, 3
Position, 3
Position (funprog), 23

Ranges, 7, 38
rank, 3, 29, 37, 42, 42, 50
rank, Vector-method, 42
rbind, 3
rbind (cbind), 10
INDEX

Reduce, 3, 23, 24
Reduce (Funprog), 23
Reduce, List-method, 24
relist, 4, 43, 43
relist, ANY, List-method, 43
rep, 44
rep.int, 3, 44
rep.int, Rle-method, 44
residuals, 4, 45, 45
residuals, PLMset-method, 45
Rle, 8, 19, 22, 40, 44, 55, 58
rownames, 3, 45, 46
rownames (rownames), 45
rownames, DataFrame-method, 46
rownames <- (rownames), 45
S3-classes-as-S4-classes, 47
S4groupGeneric, 4
sapply, 3, 33
sapply (tapply), 30
score, 4, 47
score, GenomicRanges-method, 48
score <- (score), 47
setClassUnion, 47
setdiff, 3
setdiff (sets), 48
setGeneric, 5
setMethod, 5
setOldClass, 47
sets, 48
sizeFactors (dge), 17
sizeFactors <- (dge), 17
sockconn-class
(S3-classes-as-S4-classes), 47
sort, 3, 29, 37, 42, 49, 49, 50
sort, Vector-method, 50
species, 4
species (organism_species), 38
species, AnnotationDb-method, 39
species <- (organism_species), 38
start, 3, 50, 50, 51
start, IRanges-method, 51
start <- (start), 50
strand, 4, 52
strand, GRanges-method, 53
strand <- (strand), 52
subset, 3, 53, 53, 54
subset, DataTable-method, 54
table, 3, 54, 54, 55
table, Rle-method, 55
tapply, 3, 55, 55, 56
tapply, Vector, ANY-method, 56
terminal-class
(S3-classes-as-S4-classes), 47
testPackage, 56
textConnection-class
(S3-classes-as-S4-classes), 47
union, 3, 48, 49
union (sets), 48
union, GRanges, GRanges-method, 49
unique, 4, 57, 57, 58
unique, Rle-method, 58
unlist, 4, 58, 58, 59
unlist, List-method, 59
unsplit, 59, 59
unsplit, List-method, 59
unstrand (strand), 52
unz-class (S3-classes-as-S4-classes), 47
updateObject, 4, 60
updateObject, ANY-method (updateObject), 60
updateObject, environment-method
(updateObject), 60
updateObject, envRefClass-method
(updateObject), 60
updateObject, formula-method
(updateObject), 60
updateObject, list-method
(updateObject), 60
updateObjectFromSlots (updateObject), 60
updateObjectTo, 61
url-class (S3-classes-as-S4-classes), 47
validObject, 60, 61
Vector, 6, 20, 31, 42, 50, 56
weights, 4, 62, 62
weights, PLMset-method, 62
which, 63, 63
which.max, NumericList-method, 63
which.min, 63
width, 3
width (start), 50
width <- (start), 50
xtabs, 4, 64, 64
xtabs, DataTable-method, 64