

Package ‘BiocGenerics’

April 4, 2014

Title Generic functions for Bioconductor

Description S4 generic functions needed by many Bioconductor packages.

Version 0.8.0

Author The Bioconductor Dev Team

Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>

biocViews Infrastructure

Depends methods, graphics, stats, parallel

Imports methods, graphics, stats, parallel

Suggests Biobase, IRanges, GenomicRanges, AnnotationDbi, oligoClasses,oligo, affyPLM, flow-Clust, affy, RUnit, DESeq2

License Artistic-2.0

Collate S3-classes-as-S4-classes.R append.R as.data.frame.R
as.vector.R cbind.R dge.R plotMA.R duplicated.R eval.R
Extremes.R funprog.R get.R is.unsorted.R lapply.R mapply.R
match.R nrow.R order.R paste.R rank.R rep.R row_colnames.R
sets.R sort.R table.R tapply.R unique.R unlist.R boxplot.R
image.R density.R residuals.R weights.R xtabs.R clusterApply.R
annotation.R combine.R normalize.R normarg-utils.R show-utils.R
strand.R updateObject.R update.R testPackage.R test_BiocGenerics_package.R zzz.R

R topics documented:

BiocGenerics-package	2
annotation	5
append	5
as.data.frame	6
as.vector	7
boxplot	8
cbind	9

clusterApply	10
combine	12
density	15
dge	16
duplicated	17
eval	18
evalq	19
Extremes	20
funprog	21
get	22
image	24
is.unsorted	25
lapply	26
mapply	27
match	28
normalize	29
nrow	30
order	31
paste	32
plotMA	33
rank	34
rep	35
residuals	36
row+colnames	37
S3-classes-as-S4-classes	38
sets	39
sort	40
strand	41
table	42
tapply	43
unique	44
unlist	45
updateObject	46
weights	48
xtabs	49

Index	51
--------------	-----------

BiocGenerics-package *Generic functions for Bioconductor*

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::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::lapply`, `BiocGenerics::sapply`
- `BiocGenerics::mapply`
- `BiocGenerics::match`
- `BiocGenerics::nrow`, `BiocGenerics::ncol`, `BiocGenerics::NROW`, `BiocGenerics::NCOL`
- `BiocGenerics::order`
- `BiocGenerics::paste`
- `BiocGenerics::rank`
- `BiocGenerics::rep.int`
- `BiocGenerics::rownames`, `BiocGenerics::colnames`
- sets: `BiocGenerics::union`, `BiocGenerics::intersect`, `BiocGenerics::setdiff`
- `BiocGenerics::sort`
- `BiocGenerics::table`
- `BiocGenerics::tapply`
- `BiocGenerics::unique`
- `BiocGenerics::unlist`

Generics for functions defined in package graphics:

- `BiocGenerics::boxplot`
- `BiocGenerics::image`

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`
- `normalize`
- `strand`, `strand<-`
- `updateObject`

Note

More generics can be added on request by sending an email to the Bioc-devel mailing list:

<http://bioconductor.org/help/mailling-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

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

annotation	<i>Accessing annotation information</i>
------------	---

Description

Get or set the annotation information contained in an object.

Usage

```
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

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

append	<i>Append elements to a vector-like object</i>
--------	--

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 IRanges 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

```
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

```
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,Ranges-method` and `as.data.frame,DataFrame-method` in the `IRanges` package for examples of specific `as.data.frame` methods (defined for `Ranges` and `DataFrame` objects, respectively).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

```
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	<i>Coerce an object into a vector</i>
-----------	---------------------------------------

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 `BiocGenerics` 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 `IRanges` package 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
showMethods("as.vector")
selectMethod("as.vector", "ANY") # the default method
```

boxplot

Box plots

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 `IRanges` 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

cl	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 <code>?parallel::clusterApply</code> 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

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

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

Description

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

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

Arguments

<code>x</code>	One of the values.
<code>y</code>	A second value.
<code>...</code>	Any other objects of the same class as <code>x</code> and <code>y</code> .

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 `data.frames`. 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 `data.frames` 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

- [combine,AnnotatedDataFrame,AnnotatedDataFrame-method](#), [combine,AssayData,AssayData-method](#), [combine,MIAME,MIAME-method](#), and [combine,eSet,eSet-method](#) in the Biobase package for additional combine methods.
- [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
```

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
```

dge

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., <code>CountDataSet</code> , <code>DESeqSummarizedExperiment</code> or <code>ExonCountSet</code> .
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

duplicated	<i>Determine duplicate elements</i>
------------	-------------------------------------

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.

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,Ranges-method` in the `IRanges` package for an example of a specific `duplicated` method (defined for `Ranges` 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 <code>?base::eval</code>) or by the additional methods defined in Bioconductor packages.
envir	The <i>environment</i> in which expr is to be evaluated. May be any object supported by the default method (see <code>?base::eval</code>) or by the additional methods defined in Bioconductor packages.
enclos	See <code>?base::eval</code> 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,List-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 `List` 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

Evaluate an (unevaluated) expression

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

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

Arguments

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

Value

See `?base::evalq`.

See Also

- `base::evalq` for the base `evalq` function.

Examples

```
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

```
pmax(..., na.rm=FALSE)
pmin(..., na.rm=FALSE)

pmax.int(..., na.rm=FALSE)
pmin.int(..., na.rm=FALSE)
```

Arguments

`...` One or more vector-like or matrix-like objects.
`na.rm` See `?base::pmax` for a description of this argument.

Value

See `?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

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

Examples

```

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

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

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

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

```

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 IRanges 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

```
get(x, pos=-1, envir=as.environment(pos), mode="any", inherits=TRUE)
mget(x, envir, mode="any", ifnotfound, inherits=FALSE)
```

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

```
get # note the dispatch on the x, pos and envir args only
showMethods("get")
selectMethod("get", 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
```

image

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
```

is.unsorted	<i>Test if an Object is Not Sorted</i>
-------------	--

Description

Test if an 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 not supported by the default method.

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.

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 DEVELOPPERS:

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::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,Rle-method` in the `IRanges` package for an example of a specific `is.unsorted` method (defined for `Rle` 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	<i>Apply a function over a list-like or vector-like object</i>
--------	--

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 Bioc-Generics 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 `IRanges` 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

```

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

See `?base::mapply` for a description of these arguments.

... 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.
- `mapply,List-method` in the IRanges package for an example of a specific mapply method (defined for `List` objects).
- `BiocGenerics` for a summary of all the generics defined in the BiocGenerics package.

Examples

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

match	<i>Value matching</i>
-------	-----------------------

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

```
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 `IRanges` 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.

Examples

```
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

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

Arguments

<code>object</code>	A data object, typically containing microarray data.
<code>...</code>	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

```
normalize
showMethods("normalize")

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

nrow	<i>The number of rows/columns of an array-like object</i>
------	---

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 `IRanges` 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

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

order	<i>Ordering permutation</i>
-------	-----------------------------

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

```
order(..., na.last=TRUE, decreasing=FALSE)
```

Arguments

... One or more vector-like objects, all of the same length.
na.last, decreasing 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 DEVELOPPERS:

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

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

paste

Concatenate strings

Description

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

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

Usage

```
paste(..., sep=" ", collapse=NULL)
```

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 `IRanges` 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

```
paste
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

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

```
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

```
rank(x, na.last=TRUE,
     ties.method=c("average", "first", "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 DEVELOPPERS:

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,Ranges-method` in the `IRanges` package for an example of a specific rank method (defined for `Ranges` objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

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

 rep

Replicate elements of a vector-like object

Description

`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

```
## Unlike the standard rep.int() function defined in base (default method),
## the generic function described here have a ... argument (instead of
## times).
rep.int(x, ...)
```

Arguments

<code>x</code>	The object to replicate (typically vector-like).
<code>...</code>	Additional arguments, for use in specific <code>rep.int</code> methods.

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 `IRanges` 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	<i>Row and column names</i>
--------------	-----------------------------

Description

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

NOTE: This man page is for the `rownames` 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")
colnames(x, do.NULL=TRUE, prefix="col")
```

Arguments

`x` A matrix-like object.

`do.NULL, prefix`

See `?base::rownames` for a description of these arguments.

Value

NULL or a character vector of length `nrow(x)` for `rownames` and `ncol(x)` for `colnames(x)`. See `?base::rownames` for more information about the default methods.

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

See Also

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

Examples

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

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

S3-classes-as-S4-classes

S3 classes as S4 classes

Description

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.

Details

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`

See Also

`setOldClass` and `setClassUnion` in the `methods` package.

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

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

Arguments

<code>x</code> , <code>y</code>	Vector-like objects (typically of the same class, but not necessarily).
<code>...</code>	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 BiocGenerics 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

```
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

```
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 DEVELOPPERS:

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 `IRanges` 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
```

strand	<i>Accessing strand information</i>
--------	-------------------------------------

Description

Get or set the strand information contained in an object.

Usage

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

Arguments

<code>x</code>	An object containing strand information.
<code>...</code>	Additional arguments, for use in specific methods.
<code>value</code>	The strand information to set on <code>x</code> .

Note

All the `strand` methods defined in the `GenomicRanges` package use the same set of 3 values (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.

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")

library(GenomicRanges)
showMethods("strand")
selectMethod("strand", "missing")
strand()
```

table

Cross tabulation and table creation

Description

table uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

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

Usage

```
table(...)
```

Arguments

... One or more objects which can be interpreted as factors (including character strings), or a list (or data frame) whose components can be so interpreted.

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 IRanges 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
```

tapply

Apply a function over a ragged array

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 methods.

Usage

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

Arguments

X A list-like or vector-like object.
INDEX, 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 `tapply` method.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `tapply, Vector-method` in the `IRanges` package for an example of a specific `tapply` method (defined for `Vector` objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

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

unique	<i>Extract unique elements</i>
--------	--------------------------------

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

```
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 `IRanges` 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	<i>Flatten list-like objects</i>
--------	----------------------------------

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

<code>x</code>	A list-like object.
<code>recursive</code> , <code>use.names</code>	See <code>?base::unlist</code> 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 IRanges 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
```

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 <code>message</code> 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]], ..., v`

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

`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. The corresponding elements in `object` are then updated (with `updateObject(elt, ..., verbose=ver` and used as arguments to a call to `new(class, ...)`, with ... replaced by slots from the original object. If this fails, `updateObjectFromSlots` then 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

```

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
  }
)

a <- new("UpdtA", x=1:10)
## See steps involved
updateObject(a)

removeMethod("updateObject", "UpdtA")
removeClass("UpdtA")

```

weights

Extract model 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 BiocGenerics 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
```

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 `IRanges` 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, 38
- *Topic **manip**
 - dge, 16
- *Topic **methods**
 - annotation, 5
 - append, 5
 - as.data.frame, 6
 - as.vector, 7
 - boxplot, 8
 - cbind, 9
 - clusterApply, 10
 - combine, 12
 - density, 15
 - duplicated, 17
 - eval, 18
 - Extremes, 20
 - funprog, 21
 - get, 22
 - image, 24
 - is.unsorted, 25
 - lapply, 26
 - mapply, 27
 - match, 28
 - normalize, 29
 - nrow, 30
 - order, 31
 - paste, 32
 - plotMA, 33
 - rank, 34
 - rep, 35
 - residuals, 36
 - row+colnames, 37
 - sets, 39
 - sort, 40
 - strand, 41
 - table, 42
 - tapply, 43
 - unique, 44
 - unlist, 45
 - updateObject, 46
 - weights, 48
 - xtabs, 49
- *Topic **package**
 - BiocGenerics-package, 2
 - <=, 25, 32, 35, 41
- AffyBatch, 29
- AnnotatedDataFrame, 14
- annotation, 4, 5
- annotation, eSet-method, 5
- annotation<- (annotation), 5
- anyDuplicated, 3
- anyDuplicated (duplicated), 17
- append, 3, 5, 5, 6
- append, Vector, Vector-method, 6
- as.data.frame, 3, 6, 6, 7
- as.data.frame, DataFrame-method, 7
- as.data.frame, Ranges-method, 7
- as.vector, 3, 7, 7, 8
- as.vector, AtomicList-method, 8
- as.vector, Rle-method, 8
- AsIs-class (S3-classes-as-S4-classes), 38
- AssayData, 14
- AtomicList, 8
- Bimap, 23
- BiocGenerics, 5–11, 14, 15, 17, 19, 20, 22–27, 29, 30, 32–38, 40–47, 49, 50
- BiocGenerics (BiocGenerics-package), 2
- BiocGenerics-package, 2
- boxplot, 3, 8, 8, 9
- boxplot, FeatureSet-method, 9
- bzfile-class (S3-classes-as-S4-classes), 38
- c, 45
- cbind, 3, 4, 9, 9, 10

- cbind, DataFrame-method, 10
- characterORconnection-class
 - (S3-classes-as-S4-classes), 38
- clusterApply, 4, 10, 10, 11
- clusterApplyLB, 4
- clusterApplyLB (clusterApply), 10
- clusterCall, 4
- clusterCall (clusterApply), 10
- clusterEvalQ, 4
- clusterEvalQ (clusterApply), 10
- clusterExport, 4
- clusterExport (clusterApply), 10
- clusterMap, 4
- clusterMap (clusterApply), 10
- clusterSplit, 4
- clusterSplit (clusterApply), 10
- colnames, 3
- colnames (row+colnames), 37
- combine, 4, 12
- combine, AnnotatedDataFrame, AnnotatedDataFrame-method, 14
- combine, ANY, missing-method (combine), 12
- combine, AssayData, AssayData-method, 14
- combine, data.frame, data.frame-method (combine), 12
- combine, eSet, eSet-method, 14
- combine, matrix, matrix-method (combine), 12
- combine, MIAME, MIAME-method, 14
- conditions (dge), 16
- conditions<- (dge), 16
- connection-class
 - (S3-classes-as-S4-classes), 38
- counts (dge), 16
- counts<- (dge), 16

- DataFrame, 7, 10, 30, 38
- DataTable, 50
- density, 3, 15, 15
- density, flowClust-method, 15
- design (dge), 16
- design<- (dge), 16
- dge, 16
- dim, 4
- dispTable (dge), 16
- dispTable<- (dge), 16
- duplicated, 3, 17, 17, 44, 45
- duplicated, Ranges-method, 17

- eSet, 5, 14
- estimateDispersions (dge), 16
- estimateSizeFactors (dge), 16
- eval, 3, 18, 18, 19
- eval, expression, List-method, 19
- evalq, 19, 19
- expression, 19
- Extremes, 20

- factor, 13
- FeatureSet, 9, 24, 29
- fifo-class (S3-classes-as-S4-classes), 38
- file-class (S3-classes-as-S4-classes), 38
- Filter, 3
- Filter (funprog), 21
- Find, 3
- Find (funprog), 21
- flowClust, 15
- funprog, 21

- get, 3, 22, 22, 23
- get, ANY, Bimap, missing-method, 23
- getObjectSlots (updateObject), 46
- GRanges, 39, 40, 42
- groupGeneric, 4
- gzcon-class (S3-classes-as-S4-classes), 38
- gzfile-class
 - (S3-classes-as-S4-classes), 38

- Hits, 29

- image, 3, 24, 24
- image, FeatureSet-method, 24
- InternalMethods, 4
- intersect, 3
- intersect (sets), 39
- is.unsorted, 25, 25
- is.unsorted, Rle-method, 25

- lapply, 3, 26, 26
- lapply, List-method, 26
- length, 4
- List, 19, 22, 26, 27, 46

- Map, 3
- Map (funprog), 21
- mapply, 3, 27, 27

- mapply, List-method, 27
- match, 3, 28, 28, 29
- match, Hits, Hits-method, 29
- Math, 4
- merge, 14
- message, 46
- mget, 3
- mget (get), 22
- MIAME, 14

- NCOL, 3
- NCOL (nrow), 30
- ncol, 3, 37
- ncol (nrow), 30
- normalize, 4, 29
- normalize, AffyBatch-method, 29
- normalize, FeatureSet-method, 29
- NROW, 3
- NROW (nrow), 30
- nrow, 3, 30, 30, 37
- nrow, DataFrame-method, 30

- Ops, 4
- order, 3, 25, 31, 31, 32, 35, 41
- order, Ranges-method, 32

- parApply, 4
- parApply (clusterApply), 10
- parCapply, 4
- parCapply (clusterApply), 10
- parLapply, 4
- parLapply (clusterApply), 10
- parLapplyLB, 4
- parLapplyLB (clusterApply), 10
- parRapply, 4
- parRapply (clusterApply), 10
- parSapply, 4
- parSapply (clusterApply), 10
- parSapplyLB, 4
- parSapplyLB (clusterApply), 10
- paste, 3, 32, 32, 33
- paste, Rle-method, 33
- pipe-class (S3-classes-as-S4-classes), 38
- PLMset, 37, 49
- plotDispEsts (dge), 16
- plotMA, 33, 34
- plotMA, ANY-method (plotMA), 33
- pmax, 3, 20
- pmax (Extremes), 20
- pmax, Rle-method, 20
- pmax.int, 3
- pmin, 3
- pmin (Extremes), 20
- pmin.int, 3
- Position, 3
- Position (funprog), 21

- Ranges, 7, 17, 32, 35
- rank, 3, 25, 32, 34, 34, 35, 41
- rank, Ranges-method, 35
- rbind, 3
- rbind (cbind), 9
- Reduce, 3, 21, 22
- Reduce (funprog), 21
- Reduce, List-method, 22
- rep, 35
- rep.int, 3, 35, 36
- rep.int, Rle-method, 36
- residuals, 3, 36, 36, 37
- residuals, PLMset-method, 37
- Rle, 8, 20, 25, 33, 36, 43, 45
- row+colnames, 37
- rownames, 3, 37, 38
- rownames (row+colnames), 37
- rownames, DataFrame-method, 38

- S3-classes-as-S4-classes, 38
- S4groupGeneric, 4
- sapply, 3, 27
- sapply (lapply), 26
- selectMethod, 4–11, 14, 15, 17, 19, 20, 22–27, 29, 30, 32–39, 41–47, 49, 50
- setClassUnion, 38
- setdiff, 3
- setdiff (sets), 39
- setGeneric, 4
- setMethod, 4
- setOldClass, 38
- sets, 39
- showMethods, 4–11, 14, 15, 17, 19, 20, 22–27, 29, 30, 32–39, 41–47, 49, 50
- sizeFactors (dge), 16
- sizeFactors<- (dge), 16
- sockconn-class (S3-classes-as-S4-classes), 38
- sort, 3, 25, 32, 35, 40, 40, 41
- sort, Vector-method, 41

strand, 4, 41
strand, GRanges-method, 42
strand<- (strand), 41

table, 3, 42, 42, 43
table, Rle-method, 43
tapply, 3, 43, 43, 44
tapply, Vector-method, 44
terminal-class
 (S3-classes-as-S4-classes), 38
textConnection-class
 (S3-classes-as-S4-classes), 38

union, 3, 39
union (sets), 39
union, GRanges, GRanges-method, 40
unique, 3, 44, 44, 45
unique, Rle-method, 45
unlist, 3, 45, 45, 46
unlist, List-method, 46
unz-class (S3-classes-as-S4-classes), 38
updateObject, 4, 46
updateObject, ANY-method (updateObject),
 46
updateObject, environment-method
 (updateObject), 46
updateObject, list-method
 (updateObject), 46
updateObjectFromSlots (updateObject), 46
updateObjectTo, 47
url-class (S3-classes-as-S4-classes), 38

validObject, 47
Vector, 6, 41, 44

weights, 3, 48, 48, 49
weights, PLMset-method, 49

xtabs, 3, 49, 49, 50
xtabs, DataTable-method, 50