

Package ‘BiocGenerics’

October 16, 2020

Title S4 generic functions used in Bioconductor

Description The package defines S4 generic functions used in Bioconductor.

Version 0.34.0

Author The Bioconductor Dev Team

Maintainer Bioconductor Package Maintainer <maintainer@bioconductor.org>

biocViews Infrastructure

Depends R (>= 3.6.0), methods, utils, graphics, stats, parallel

Imports methods, utils, graphics, stats, parallel

Suggests Biobase, S4Vectors, IRanges, GenomicRanges, DelayedArray,
Biostrings, Rsamtools, AnnotationDbi, oligoClasses, oligo,
affyPLM, flowClust, affy, DESeq2, MSnbase, annotate, RUnit

License Artistic-2.0

Collate S3-classes-as-S4-classes.R utils.R normarg-utils.R
hotfix73465.R replaceSlots.R append.R as.data.frame.R as.list.R
as.vector.R cbind.R colSums.R do.call.R duplicated.R eval.R
Extremes.R funprog.R get.R grep.R is.unsorted.R lapply.R
mapply.R match.R mean.R nrow.R order.R paste.R rank.R rep.R
row_colnames.R sets.R sort.R start.R subset.R t.R table.R
tapply.R unique.R unlist.R unsplit.R relist.R var.R which.R
boxplot.R image.R density.R IQR.R mad.R residuals.R weights.R
xtabs.R clusterApply.R annotation.R combine.R dbconn.R dge.R
dims.R fileName.R normalize.R Ontology.R organism_species.R
path.R plotMA.R plotPCA.R score.R strand.R toTable.R type.R
updateObject.R testPackage.R zzz.R

git_url <https://git.bioconductor.org/packages/BiocGenerics>

git_branch RELEASE_3_11

git_last_commit f7c2020

git_last_commit_date 2020-04-27

Date/Publication 2020-10-16

R topics documented:

BiocGenerics-package	3
annotation	5
append	6

as.data.frame	7
as.vector	8
boxplot	9
cbind	10
clusterApply	11
colSums	13
combine	14
dbconn	16
density	17
dge	18
dims	18
do.call	19
duplicated	20
eval	21
evalq	22
Extremes	23
fileName	24
funprog	25
get	26
grep	27
image	28
IQR	29
is.unsorted	30
lapply	31
mad	32
mapply	33
match	34
mean	35
normalize	36
nrow	37
Ontology	38
order	38
organism_species	39
paste	41
path	42
plotMA	44
plotPCA	45
rank	46
relist	47
rep	48
residuals	49
row+colnames	50
S3-classes-as-S4-classes	51
score	52
sets	53
sort	54
start	55
strand	57
subset	58
t	59
table	60
tapply	61

testPackage	62
toTable	63
type	64
unique	65
unlist	66
unsplit	67
updateObject	68
var	70
weights	71
which	72
xtabs	73

Index	75
--------------	-----------

BiocGenerics-package *S4 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::rbind`, `BiocGenerics::cbind`
- `BiocGenerics::colSums`, `BiocGenerics::rowSums`, `BiocGenerics::colMeans`, `BiocGenerics::rowMeans`
- `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::mapply`
- `BiocGenerics::match`, `BiocGenerics::%in%`
- `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::pos
- BiocGenerics::subset
- BiocGenerics::table
- BiocGenerics::tapply
- BiocGenerics::unique
- BiocGenerics::unlist

Generics for functions defined in package **utils**:

- BiocGenerics::relist

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
- dbconn, dbfile
- dims
- fileName
- normalize
- Ontology
- organism, organism<-, species, species<-
- path, path<-, basename, basename<-, dirname, dirname<-
- plotMA
- plotPCA
- score, score<-
- strand, strand<-, invertStrand
- toTable
- type, type<-
- 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

Accessing annotation information

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

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

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

as.data.frame	<i>Coerce an object into a data frame</i>
---------------	---

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.
- `toTable` for an alternative to `as.data.frame`.
- `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, IntegerRanges-method` in the **IRanges** package, for examples of specific `as.data.frame` methods (defined for `DataFrame` and `IntegerRanges` 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

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 **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 <code>?base::as.vector</code> 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
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

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

NOTE: This man page is for the `rbind` and `cbind` *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

```
rbind(..., deparse.level=1)
cbind(..., 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 `rbind` and `cbind` methods.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `rbind,DataTable-method` and `cbind,DataFrame-method` in the **S4Vectors** package for examples of specific `rbind` and `cbind` methods (defined for `DataTable` and `DataFrame` objects, respectively).
- **BiocGenerics** for a summary of all the generics defined in the **BiocGenerics** package.

Examples

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

cbind # note the dispatch on the '...' arg only
showMethods("cbind")
selectMethod("cbind", "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, ..., chunk.size=NULL)
parSapply(cl=NULL, X, FUN, ..., simplify=TRUE,
          USE.NAMES=TRUE, chunk.size=NULL)
parApply(cl=NULL, X, MARGIN, FUN, ..., chunk.size=NULL)
parRapply(cl=NULL, x, FUN, ..., chunk.size=NULL)
parCapply(cl=NULL, x, FUN, ..., chunk.size=NULL)

parLapplyLB(cl=NULL, X, fun, ..., chunk.size=NULL)
parSapplyLB(cl=NULL, X, FUN, ..., simplify=TRUE,
            USE.NAMES=TRUE, chunk.size=NULL)
```

Arguments

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

```

colSums

Form Row and Column Sums and Means

Description

Form row and column sums and means for rectangular objects..

NOTE: This man page is for the rowSums, colSums, rowMeans, and colMeans *S4 generic functions* defined in the **BiocGenerics** package. See `?base::colSums` for the default methods (defined in the **base** package). Bioconductor packages can define specific methods for objects (typically array-like) not supported by the default methods.

Usage

```

colSums(x, na.rm=FALSE, dims=1, ...)
rowSums(x, na.rm=FALSE, dims=1, ...)
colMeans(x, na.rm=FALSE, dims=1, ...)
rowMeans(x, na.rm=FALSE, dims=1, ...)

```

Arguments

x	A rectangular object, like a matrix or data frame.
na.rm, dims	See <code>?base::colSums</code> for a description of these arguments.
...	Additional arguments, for use in specific methods.

Value

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

See Also

- `base::colSums` for the default colSums, rowSums, colMeans, and colSums methods.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `colSums,DelayedMatrix-method` in the **DelayedArray** package for an example of a specific colSums method (defined for `DelayedMatrix` objects).
- **BiocGenerics** for a summary of all the generics defined in the **BiocGenerics** package.

Examples

```

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

```

`combine`*Combining or merging different Bioconductor data structures*

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
```

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
```

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

dims *Get the dimensions of all elements in a list-like object*

Description

Get the dimensions of all elements in a list-like object.

Usage

```
dims(x)
```

Arguments

x List-like object (or environment) containing one or several array-like objects with the same number of dimensions.

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.
- [dims,eSet-method](#) in the **Biobase** package for an example of a specific dims method (defined for [eSet](#) objects).
- [BiocGenerics](#) for a summary of all the generics defined in the **BiocGenerics** package.

Examples

```
dims
showMethods("dims")

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

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.

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

<code>expr</code>	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.
<code>envir</code>	The <i>environment</i> in which <code>expr</code> 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.
<code>enclos</code>	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,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

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

<code>...</code>	One or more vector-like or matrix-like objects.
<code>na.rm</code>	See <code>?base:pmax</code> 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 **S4Vectors** 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
```

fileName	<i>Accessing the file name of an object</i>
----------	---

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")
```


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 <code>?base::Reduce</code> 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

```
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
```

 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 `grep1` *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

```
grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
      fixed = FALSE, useBytes = FALSE, invert = FALSE)
grep1(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 `grep1` 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

```
grep # note the dispatch on 'pattern' and 'x' args only
showMethods("grep")
selectMethod("grep", "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
```

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.

Usage

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

Arguments

<code>x</code>	A vector-like object.
<code>na.rm, strictly</code>	See <code>?base::is.unsorted</code> for a description of these arguments.
<code>...</code>	Additional arguments, for use in specific methods. Note that <code>base::is.unsorted</code> (the default method) only takes the <code>x</code> , <code>na.rm</code> , and <code>strictly</code> 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	<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=TRUE)` 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
```

`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

```
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.

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`

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.
- [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

Value matching

Description

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

%in% is a binary operator that returns a logical vector of the length of its left operand indicating if the elements in it have a match or not.

NOTE: This man page is for the match and %in% *S4 generic functions* defined in the **BiocGenerics** package. See `?base::match` 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

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

```
x %in% table
```

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 methods (see `?base::match` for the value returned by the default methods).

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

Note

The default `base::match` 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 and %in% methods.
- `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` and `%in%,Rle,ANY-method` in the **S4Vectors** package for examples of specific match and %in% methods (defined for **Hits** and **Rle** objects, respectively).
- **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

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

```

mean

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

NOTE: This man page is for the mean *S4 generic function* defined in the **BiocGenerics** package. See `?base:mean` 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

```
mean(x, ...)
```

Arguments

```

x           typically a vector-like object
...        see mean

```

Value

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

Examples

```

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

```

normalize	<i>Normalize an object</i>
-----------	----------------------------

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

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

```
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 **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
```

```

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

```

Ontology	<i>Generic Ontology getter</i>
----------	--------------------------------

Description

Get the Ontology of an object.

Usage

```
Ontology(object)
```

Arguments

object An object with an Ontology.

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.
- [Ontology, GOTerms-method](#) in the **AnnotationDbi** package for an example of a specific Ontology method (defined for [GOTerms](#) objects).
- [BiocGenerics](#) for a summary of all the generics defined in the **BiocGenerics** package.

Examples

```

Ontology
showMethods("Ontology")

library(AnnotationDbi)
showMethods("Ontology")
selectMethod("Ontology", "GOTerms")

```

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, method=c("auto", "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,IntegerRanges-method` in the **IRanges** package for an example of a specific order method (defined for `IntegerRanges` objects).
- `BiocGenerics` for a summary of all the generics defined in the **BiocGenerics** package.

Examples

```
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

```
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 sub-species) 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.

See Also

- http://bioconductor.org/packages/release/BiocViews.html#___Organism for browsing the annotation packages currently available in Bioconductor by organism.
- [showMethods](#) for displaying a summary of the methods defined for a given generic function.
- [selectMethod](#) for getting the definition of a specific method.
- [organism,character-method](#) and [organism,chromLocation-method](#) in the **annotate** package for examples of specific organism methods (defined for [character](#) and [chromLocation](#) objects).
- [species,AnnotationDb-method](#) in the **AnnotationDbi** package for an example of a specific species method (defined for [AnnotationDb](#) objects).
- [BiocGenerics](#) for a summary of all the generics defined in the **BiocGenerics** package.

Examples

```
## organism() getter:
organism
showMethods("organism")

library(annotate)
showMethods("organism")
selectMethod("organism", "character")
selectMethod("organism", "chromLocation")

## organism() setter:
```



```
`organism<-`  
showMethods("organism<-")  
  
## species() getter:  
species  
showMethods("species")  
  
library(AnnotationDbi)  
selectMethod("species", "AnnotationDb")  
  
## species() setter:  
`species<-`  
showMethods("species<-")
```

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 **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

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

path

Accessing the path of an object

Description

Get or set the path of an object.

Usage

```
path(object, ...)
path(object, ...) <- value

basename(path, ...)
basename(path, ...) <- value

dirname(path, ...)
dirname(path, ...) <- value

## The purpose of the following methods is to make the basename() and
## dirname() getters work out-of-the-box on any object for which the
## path() getter works.

## S4 method for signature 'ANY'
basename(path, ...)

## S4 method for signature 'ANY'
dirname(path, ...)

## The purpose of the following replacement methods is to make the
## basename() and dirname() setters work out-of-the-box on any object
## for which the path() getter and setter work.

## S4 replacement method for signature 'character'
basename(path, ...) <- value

## S4 replacement method for signature 'ANY'
basename(path, ...) <- value

## S4 replacement method for signature 'character'
dirname(path, ...) <- value

## S4 replacement method for signature 'ANY'
dirname(path, ...) <- value
```

Arguments

object	An object containing paths. Even though it will typically contain a single path, object can actually contain an arbitrary number of paths.
...	Additional arguments, for use in specific methods.
value	For path<-, the paths to set on object. For basename<- or dirname<-, the basenames or dirnames to set on path.
path	A character vector <i>or an object containing paths</i> .

Value

A character vector for path(object), basename(path), and dirname(path). Typically of length 1 but not necessarily. Possibly with names on it for path(object).

See Also

- base: [:basename](#) for the functions the basename and dirname generics are based on.
- [showMethods](#) for displaying a summary of the methods defined for a given generic function.
- [selectMethod](#) for getting the definition of a specific method.
- [path, RsamtoolsFile-method](#) in the **Rsamtools** package for an example of a specific path method (defined for [RsamtoolsFile](#) objects).
- [BiocGenerics](#) for a summary of all the generics defined in the **BiocGenerics** package.

Examples

```
## -----
## GENERIC FUNCTIONS AND DEFAULT METHODS
## -----

path
showMethods("path")

`path<-`
showMethods("path<-")

basename
showMethods("basename")

`basename<-`
showMethods("basename<-")

dirname
showMethods("dirname")

`dirname`
showMethods("dirname<-")

## Default basename() and dirname() getters:
selectMethod("basename", "ANY")
selectMethod("dirname", "ANY")

## Default basename() and dirname() setters:
selectMethod("basename<-", "character")
```

```

selectMethod("basename<-", "ANY")
selectMethod("dirname<-", "character")
selectMethod("dirname<-", "ANY")

## -----
## OBJECTS CONTAINING PATHS
## -----

## Let's define a simple class to represent objects that contain paths:
setClass("A", slots=c(stuff="ANY", path="character"))

a <- new("A", stuff=runif(5),
         path=c(one="path/to/file1", two="path/to/file2"))

## path() getter:
setMethod("path", "A", function(object) object@path)

path(a)

## Because the path() getter works on 'a', now the basename() and
## dirname() getters also work:
basename(a)
dirname(a)

## path() setter:
setReplaceMethod("path", "A",
  function(object, ..., value)
  {
    if (length(list(...)) != 0L) {
      dots <- match.call(expand.dots=FALSE)[[3L]]
      stop(BiocGenerics:::unused_arguments_msg(dots))
    }
    object@path <- value
    object
  }
)

a <- new("A", stuff=runif(5))
path(a) <- c(one="path/to/file1", two="path/to/file2")
path(a)

## Because the path() getter and setter work on 'a', now the basename()
## and dirname() setters also work:
basename(a) <- toupper(basename(a))
path(a)
dirname(a) <- "~/MyDataFiles"
path(a)

```

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

```
showMethods("plotMA")

suppressWarnings(
  if(require("DESeq2"))
    example("plotMA", package="DESeq2", local=TRUE)
)
```

plotPCA

PCA-plot: Principal Component Analysis plot

Description

A generic function which produces a PCA-plot.

Usage

```
plotPCA(object, ...)
```

Arguments

object A data object, typically containing gene expression information.

... 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.
- [plotPCA](#) in the **DESeq2** package for an example method that uses this generic.
- [BiocGenerics](#) for a summary of all the generics defined in the **BiocGenerics** package.

Examples

```
showMethods("plotPCA")

suppressWarnings(
  if(require("DESeq2"))
    example("plotPCA", 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", "last", "random", "max", "min"),
     ...)
```

Arguments

`x` A vector-like object.

`na.last, ties.method` See `?base::rank` for a description of these arguments.

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

Note that `base::rank` (the default method) only takes the `x`, `na.last`, and `ties.method` 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

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

relist	<i>Re-listing an unlist()ed object</i>
--------	--

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 <code>skeleton</code> 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	<i>Replicate elements of a vector-like object</i>
-----	---

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

```
rep.int(x, times)
```

Arguments

<code>x</code>	The object to replicate (typically vector-like).
<code>times</code>	See <code>?base::rep.int</code> 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

<code>x</code>	A matrix-like object.
<code>do.NULL</code> , <code>prefix</code>	See <code>?base::rownames</code> for a description of these arguments.
<code>value</code>	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 `nrow(x)` for `rownames` and length `ncol(x)` for `colnames(x)`.

See `?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

- `base::rownames` for the default `rownames`, ``rownames<-``, `colnames`, 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 **S4Vectors** 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() getter:
rownames # note the dispatch on the 'x' arg only
showMethods("rownames")
selectMethod("rownames", "ANY") # the default method

## rownames() setter:
`rownames<-`
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:
`colnames<-`
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 **Bioc-Generics** 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](#), [dist](#)

See Also

[setOldClass](#) and [setClassUnion](#) in the **methods** package.

score	<i>Score accessor</i>
-------	-----------------------

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

```
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

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

Arguments

<code>x, 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 **GenomicRanges** 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,GenomicRanges,GenomicRanges-method` in the **GenomicRanges** package for examples of specific union, intersect, and `setdiff` methods (defined for **GenomicRanges** 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	<i>Sorting a vector-like object</i>
------	-------------------------------------

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 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

The start(), end(), width(), and pos() generic getters and setters

Description

Get or set the start, end, width, or single positions stored in an object.

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

Usage

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

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

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

pos(x)
```

Arguments

<code>x</code>	For the <code>start()</code> , <code>end()</code> , and <code>width()</code> getters/setters: an object containing start, end, and width values. For the <code>pos{}</code> getter: an object containing single positions.
<code>...</code>	Additional arguments, for use in specific methods.
<code>value</code>	The start, end, or width values to set on <code>x</code> .

Value

See specific methods defined in Bioconductor packages.

See Also

- `stats::start` in the **stats** package for the start and end S3 generics.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `start,IRanges-method` in the **IRanges** package for examples of specific start, end, and width methods (defined for **IRanges** objects).
- `pos,UnstitchedIPos-method` in the **IRanges** package for an example of a specific pos method (defined for **UnstitchedIPos** objects).
- **BiocGenerics** for a summary of all the generics defined in the **BiocGenerics** package.

Examples

```
## 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

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

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

## pos() getter:
pos
showMethods("pos")
selectMethod("pos", "UnstitchedIPos") # pos() getter for UnstitchedIPos
# objects
```

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

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

Value

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<-")
```

subset	<i>Subsetting vector-like, matrix-like and data-frame-like objects</i>
--------	--

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 <code>?base::subset</code> 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

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

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

t

*Matrix Transpose***Description**

Given a rectangular object `x`, `t` returns the transpose of `x`.

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

Usage

```
t(x)
```

Arguments

`x` a rectangular object, like a matrix or data frame

Value

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

Examples

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

table	<i>Cross tabulation and table creation</i>
-------	--

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 **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
```

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 method.

Usage

```
tapply(X, INDEX, FUN=NULL, ..., default=NA, 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, ..., default, 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, ANY-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' and 'INDEX' args only
showMethods("tapply")
selectMethod("tapply", c("ANY", "ANY")) # the default method
```

testPackage	<i>Run RUnit package unit tests</i>
-------------	-------------------------------------

Description

testPackage helps developers implement unit tests using the **RUnit** testing conventions.

Usage

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

Arguments

pkgname	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 path.
subdir	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 inst/ below the root.
pattern	A character(1) regular expression describing the file names to be evaluated; typically used to restrict tests to a subset of all test files.
path	A character(1) directory path indicating, when pkgname is missing or NULL, where unit tests will be searched. path can be any location at or below the package root.

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

```
## 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)
```

toTable	<i>An alternative to <code>as.data.frame()</code></i>
---------	---

Description

toTable() is an *S4 generic function* provided as an alternative to `as.data.frame()`.

Usage

```
toTable(x, ...)
```

Arguments

x	The object to turn into a data frame.
...	Additional arguments, for use in specific methods.

Value

A data frame.

See Also

- The `as.data.frame` *S4 generic* defined in the **BiocGenerics** package.
- `showMethods` for displaying a summary of the methods defined for a given generic function.
- `selectMethod` for getting the definition of a specific method.
- `toTable,Bimap-method` in the **AnnotationDbi** package for an example of a specific toTable method (defined for **Bimap** objects).
- **BiocGenerics** for a summary of all the generics defined in the **BiocGenerics** package.

Examples

```
toTable
showMethods("toTable")

library(AnnotationDbi)
showMethods("toTable")
selectMethod("toTable", "Bimap")
```

type *Accessing the type of an object*

Description

Get or set the *type* of an object.

Note that `type` and `type<-` are defined as *S4 generic functions* and what *type* means exactly (and what `type()` returns) depends on the objects for which `type` and/or `type<-` methods are defined.

Usage

```
type(x)
type(x) <- value
```

Arguments

x	Any object for which the <code>type()</code> getter or setter is defined. Note that objects will either: not support the getter or setter at all, or support only the getter, or support the getter and setter.
value	The type to set on x (assuming x supports the <code>type()</code> setter). <code>value</code> is typically (but not necessarily) expected to be a single string (i.e. a character vector of length 1).

Value

`type(x)` returns the type of x, typically (but not necessarily) as a single string (i.e. as a character vector of length 1).

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.
- [type,ANY-method](#) in the **DelayedArray** package for the default `type` method.
- [type,PairwiseAlignments-method](#) in the **Biostrings** package for an example of a specific `type` method (defined for [PairwiseAlignments](#) objects).
- [BiocGenerics](#) for a summary of all the generics defined in the **BiocGenerics** package.

Examples

```
type
showMethods("type")

`type<-`
showMethods("type<-")

library(DelayedArray)
showMethods("type")
selectMethod("type", "ANY") # the default "type" method

library(Biostrings)
showMethods("type")
selectMethod("type", "PairwiseAlignments")
```

`unique`*Extract unique elements*

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 **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

<code>value</code>	A list-like object.
<code>f</code>	A factor or other grouping object that corresponds to the <code>f</code> symbol in <code>value <-split(x, f)</code> .
<code>drop</code>	See <code>?base::unsplit</code> 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	<i>Update an object to its current class definition</i>
--------------	---

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]], ... ,

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

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=)` 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

```
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")
```

var

Variance and Standard Deviation

Description

`var` and `sd` compute the variance and standard deviation of a vector `x`.

NOTE: This man page is for the `var` and `sd`, *S4 generic functions* defined in the **BiocGenerics** package. See `?stats::var` and `?stats::sd` for the default methods (defined in the **stats** package). Bioconductor packages can define specific methods for objects (typically array-like) not supported by the default method.

Usage

```
var(x, y = NULL, na.rm = FALSE, use)
sd(x, na.rm = FALSE)
```

Arguments

<code>x</code>	a vector-like object
<code>y</code>	a vector-like object, or <code>NULL</code>
<code>na.rm</code> , <code>use</code>	see var

Value

See `?stats::var` and `?stats::sd` 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 object.

See Also

- `stats::var` and `stats::sd` 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

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

weights	<i>Extract model weights</i>
---------	------------------------------

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
```

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, addNA=FALSE, exclude=if(!addNA)c(NA, NaN),
      drop.unused.levels=FALSE)
```

Arguments

formula, subset, sparse, na.action, addNA, 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

- * **classes**
 - S3-classes-as-S4-classes, 51
- * **manip**
 - dge, 18
- * **methods**
 - annotation, 5
 - append, 6
 - as.data.frame, 7
 - as.vector, 8
 - boxplot, 9
 - cbind, 10
 - clusterApply, 11
 - colSums, 13
 - combine, 14
 - dbconn, 16
 - density, 17
 - dims, 18
 - do.call, 19
 - duplicated, 20
 - eval, 21
 - Extremes, 23
 - fileName, 24
 - funprog, 25
 - get, 26
 - grep, 27
 - image, 28
 - IQR, 29
 - is.unsorted, 30
 - lapply, 31
 - mad, 32
 - mapply, 33
 - match, 34
 - mean, 35
 - normalize, 36
 - nrow, 37
 - Ontology, 38
 - order, 38
 - organism_species, 39
 - paste, 41
 - path, 42
 - plotMA, 44
 - plotPCA, 45
 - rank, 46
 - relist, 47
 - rep, 48
 - residuals, 49
 - row+colnames, 50
 - score, 52
 - sets, 53
 - sort, 54
 - start, 55
 - strand, 57
 - subset, 58
 - t, 59
 - table, 60
 - tapply, 61
 - testPackage, 62
 - toTable, 63
 - type, 64
 - unique, 65
 - unlist, 66
 - unsplit, 67
 - updateObject, 68
 - var, 70
 - weights, 71
 - which, 72
 - xtabs, 73
- * **package**
 - BiocGenerics-package, 3
 - <=, 39, 47, 54
 - %in%(match), 34
 - %in%, 3
 - %in%,Rle,ANY-method, 34
 - AffyBatch, 36
 - AnnotatedDataFrame, 15
 - annotation, 4, 5
 - annotation,eSet-method, 6
 - annotation<-(annotation), 5
 - AnnotationDb, 40
 - anyDuplicated, 3
 - anyDuplicated(duplicated), 20
 - append, 3, 6, 6, 7
 - append,Vector,Vector-method, 7
 - as.data.frame, 3, 7, 7, 63
 - as.data.frame,DataFrame-method, 7
 - as.data.frame,IntegerRanges-method, 7

- as.vector, [3](#), [8](#), [8](#)
- as.vector, AtomicList-method, [8](#)
- as.vector, Rle-method, [8](#)
- AsIs, [51](#)
- AsIs-class (S3-classes-as-S4-classes), [51](#)
- AssayData, [15](#)
- AtomicList, [8](#)
- basename, [4](#), [43](#)
- basename (path), [42](#)
- basename, ANY-method (path), [42](#)
- basename<- (path), [42](#)
- basename<-, ANY-method (path), [42](#)
- basename<-, character-method (path), [42](#)
- Bimap, [27](#), [63](#)
- BiocGenerics, [6–10](#), [12](#), [13](#), [15–17](#), [19–25](#), [27–41](#), [43](#), [45–50](#), [52](#), [53](#), [55](#), [56](#), [58–61](#), [63–67](#), [69–73](#)
- BiocGenerics (BiocGenerics-package), [3](#)
- BiocGenerics-package, [3](#)
- boxplot, [4](#), [9](#), [9](#)
- boxplot, FeatureSet-method, [9](#)
- bzfile-class (S3-classes-as-S4-classes), [51](#)
- c, [66](#)
- cbind, [3](#), [5](#), [10](#), [10](#)
- cbind, DataFrame-method, [10](#)
- characterORconnection-class (S3-classes-as-S4-classes), [51](#)
- chromLocation, [40](#)
- 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](#)
- colMeans, [3](#)
- colMeans (colSums), [13](#)
- colnames, [4](#)
- colnames (row+colnames), [50](#)
- colnames<- (row+colnames), [50](#)
- colSums, [3](#), [13](#), [13](#)
- colSums, DelayedMatrix-method, [13](#)
- combine, [4](#), [14](#)
- combine, AnnotatedDataFrame, AnnotatedDataFrame-method, [15](#)
- combine, ANY, missing-method (combine), [14](#)
- combine, AssayData, AssayData-method, [15](#)
- combine, data.frame, data.frame-method (combine), [14](#)
- combine, eSet, eSet-method, [15](#)
- combine, matrix, matrix-method (combine), [14](#)
- combine, MIAME, MIAME-method, [15](#)
- conditions (dge), [18](#)
- conditions<- (dge), [18](#)
- connection, [51](#)
- connection-class (S3-classes-as-S4-classes), [51](#)
- counts (dge), [18](#)
- counts<- (dge), [18](#)
- DataFrame, [7](#), [10](#), [37](#), [50](#)
- DataTable, [10](#), [59](#), [73](#)
- dbconn, [4](#), [16](#), [16](#)
- dbconn, AnnotationDb-method, [16](#)
- dbfile, [4](#)
- dbfile (dbconn), [16](#)
- DelayedMatrix, [13](#)
- density, [4](#), [17](#), [17](#)
- density, flowClust-method, [17](#)
- design (dge), [18](#)
- design<- (dge), [18](#)
- dge, [18](#)
- dim, [5](#)
- dims, [4](#), [18](#)
- dims, eSet-method, [19](#)
- dirname, [4](#)
- dirname (path), [42](#)
- dirname, ANY-method (path), [42](#)
- dirname<- (path), [42](#)
- dirname<-, ANY-method (path), [42](#)
- dirname<-, character-method (path), [42](#)
- dispTable (dge), [18](#)
- dispTable<- (dge), [18](#)
- dist, [51](#)
- dist-class (S3-classes-as-S4-classes), [51](#)
- do.call, [3](#), [19](#), [19](#), [20](#)
- duplicated, [3](#), [20](#), [20](#), [21](#), [65](#)
- duplicated, Rle-method, [21](#)
- end, [4](#)
- end (start), [55](#)
- end<- (start), [55](#)
- eSet, [6](#), [15](#), [19](#)
- estimateDispersions (dge), [18](#)

- estimateSizeFactors (dge), 18
- eval, 3, 21, 21, 22
- eval, expression, Vector-method, 22
- evalq, 22, 22, 23
- expression, 22
- Extremes, 23
- factor, 14
- FeatureSet, 9, 28, 36
- fifo-class (S3-classes-as-S4-classes), 51
- file-class (S3-classes-as-S4-classes), 51
- fileName, 4, 24
- fileName, MScmap-method, 24
- Filter, 3
- Filter (funprog), 25
- Find, 3
- Find (funprog), 25
- flowClust, 17
- funprog, 25
- GenomicRanges, 30, 52, 53
- get, 3, 26, 26, 27
- get, ANY, Bimap, missing-method, 27
- getObjectSlots (updateObject), 68
- GOTerms, 38
- GRanges, 58
- grep, 3, 27, 27, 28
- grepl, 3
- grepl (grep), 27
- groupGeneric, 5
- gzcon-class (S3-classes-as-S4-classes), 51
- gzfile-class (S3-classes-as-S4-classes), 51
- Hits, 34, 59
- image, 4, 28, 28
- image, FeatureSet-method, 28
- IntegerRanges, 7, 39
- InternalMethods, 5
- intersect, 4
- intersect (sets), 53
- invertStrand, 4
- invertStrand (strand), 57
- invertStrand, ANY-method (strand), 57
- IQR, 29, 29
- IRanges, 56
- is.unsorted, 3, 30, 30
- is.unsorted, GenomicRanges-method, 30
- lapply, 3, 31, 31
- lapply, List-method, 31
- length, 5
- List, 25, 31, 48, 66, 67
- mad, 32, 32
- Map, 3
- Map (funprog), 25
- mapply, 3, 33, 33
- match, 3, 34, 34
- match, Hits, Hits-method, 34
- Math, 5
- mean, 35, 35
- mean, Rle-method, 35
- merge, 15
- message, 68
- mget, 3
- mget (get), 26
- MIAME, 15
- MScmap, 24
- NCOL, 3
- NCOL (nrow), 37
- ncol, 3, 50
- ncol (nrow), 37
- normalize, 4, 36
- normalize, AffyBatch-method, 36
- normalize, FeatureSet-method, 36
- NROW, 3
- NROW (nrow), 37
- nrow, 3, 37, 37, 50
- nrow, DataFrame-method, 37
- NumericList, 72
- Ontology, 4, 38
- Ontology, GOTerms-method, 38
- Ops, 5
- order, 3, 30, 38, 38, 39, 47, 54
- order, IntegerRanges-method, 39
- organism, 4
- organism (organism_species), 39
- organism, character-method, 40
- organism, chromLocation-method, 40
- organism<- (organism_species), 39
- organism_species, 39
- PairwiseAlignments, 64
- 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, 4, 41, 41
- paste, Rle-method, 41
- path, 4, 42
- path, RsamtoolsFile-method, 43
- path<- (path), 42
- pipe-class (S3-classes-as-S4-classes), 51
- PLMset, 49, 71
- plotDispEsts (dge), 18
- plotMA, 4, 44, 45
- plotMA, ANY-method (plotMA), 44
- plotPCA, 4, 45, 46
- plotPCA, ANY-method (plotPCA), 45
- pmax, 3, 23
- pmax (Extremes), 23
- pmax, Rle-method, 23
- pmax.int, 3
- pmin, 3
- pmin (Extremes), 23
- pmin.int, 3
- pos, 4
- pos (start), 55
- pos, UnstitchedIPos-method, 56
- Position, 3
- Position (funprog), 25
- rank, 4, 30, 39, 46, 46, 47, 54
- rank, Vector-method, 47
- rbind, 3
- rbind (cbind), 10
- rbind, DataTable-method, 10
- Reduce, 3, 25
- Reduce (funprog), 25
- Reduce, List-method, 25
- relist, 4, 47, 47, 48
- relist, ANY, List-method, 48
- rep, 48
- rep.int, 4, 48
- rep.int, Rle-method, 48
- residuals, 4, 49, 49
- residuals, PLMset-method, 49
- Rle, 8, 21, 23, 34, 35, 41, 48, 60, 65
- row+colnames, 50
- rowMeans, 3
- rowMeans (colSums), 13
- rownames, 4, 50
- rownames (row+colnames), 50
- rownames, DataFrame-method, 50
- rownames<- (row+colnames), 50
- rowSums, 3
- rowSums (colSums), 13
- RsamtoolsFile, 43
- S3-classes-as-S4-classes, 51
- S4groupGeneric, 5
- sapply, 3, 33
- sapply (lapply), 31
- score, 4, 52
- score, GenomicRanges-method, 52
- score<- (score), 52
- sd, 70
- sd (var), 70
- selectMethod, 5–10, 12, 13, 15–17, 19–25, 27–41, 43, 45–50, 52, 53, 55, 56, 58–61, 63–67, 69–73
- setClassUnion, 51
- setdiff, 4
- setdiff (sets), 53
- setGeneric, 5
- setMethod, 5
- setOldClass, 51
- sets, 53
- showMethods, 5–10, 12, 13, 15–17, 19–25, 27–41, 43, 45–50, 52, 53, 55, 56, 58–61, 63–67, 69–73
- sizeFactors (dge), 18
- sizeFactors<- (dge), 18
- sockconn-class (S3-classes-as-S4-classes), 51
- sort, 4, 30, 39, 47, 54, 54, 55
- sort, Vector-method, 55
- species, 4
- species (organism_species), 39
- species, AnnotationDb-method, 40
- species<- (organism_species), 39
- start, 4, 55, 55, 56
- start, IRanges-method, 56
- start<- (start), 55
- strand, 4, 57
- strand, GRanges-method, 58
- strand<- (strand), 57
- subset, 4, 58, 58, 59
- subset, DataTable-method, 59
- t, 59, 59
- t, Hits-method, 59
- table, 4, 60, 60
- table, Rle-method, 60
- tapply, 4, 61, 61

- tapply, Vector, ANY-method, [61](#)
- terminal-class
 - (S3-classes-as-S4-classes), [51](#)
- testPackage, [62](#)
- textConnection-class
 - (S3-classes-as-S4-classes), [51](#)
- toTable, [4](#), [7](#), [63](#)
- toTable, Bimap-method, [63](#)
- type, [4](#), [64](#)
- type, ANY-method, [64](#)
- type, PairwiseAlignments-method, [64](#)
- type<- (type), [64](#)

- union, [4](#), [53](#)
- union (sets), [53](#)
- union, GenomicRanges, GenomicRanges-method,
[53](#)
- unique, [4](#), [65](#), [65](#)
- unique, Rle-method, [65](#)
- unlist, [4](#), [66](#), [66](#)
- unlist, List-method, [66](#)
- unsplit, [67](#), [67](#)
- unsplit, List-method, [67](#)
- UnstitchedIPos, [56](#)
- unstrand (strand), [57](#)
- unz-class (S3-classes-as-S4-classes), [51](#)
- updateObject, [4](#), [68](#)
- updateObject, ANY-method (updateObject),
[68](#)
- updateObject, environment-method
(updateObject), [68](#)
- updateObject, envRefClass-method
(updateObject), [68](#)
- updateObject, formula-method
(updateObject), [68](#)
- updateObject, list-method
(updateObject), [68](#)
- updateObjectFromSlots (updateObject), [68](#)
- updateObjectTo, [69](#)
- url-class (S3-classes-as-S4-classes), [51](#)

- validObject, [68](#), [69](#)
- var, [70](#), [70](#)
- Vector, [7](#), [22](#), [47](#), [55](#), [61](#)

- weights, [4](#), [71](#), [71](#)
- weights, PLMset-method, [71](#)
- which, [72](#), [72](#)
- which.max, NumericList-method, [72](#)
- which.min, [72](#)
- width, [4](#)
- width (start), [55](#)
- width<- (start), [55](#)

- xtabs, [4](#), [73](#), [73](#)
- xtabs, DataTable-method, [73](#)