Package ‘BiocGenerics’

May 16, 2024

Title S4 generic functions used in Bioconductor

Description The package defines many S4 generic functions used in Bioconductor.

biocViews Infrastructure

URL https://bioconductor.org/packages/BiocGenerics

BugReports https://github.com/Bioconductor/BiocGenerics/issues

Version 0.50.0

License Artistic-2.0

Author The Bioconductor Dev Team

Maintainer Hervé Pagès <hpages.on.github@gmail.com>

Depends R (>= 4.0.0), methods, utils, graphics, stats

Imports methods, utils, graphics, stats

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


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

git_branch RELEASE_3_19

git_last_commit d23d8dd
Contents

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BiocGenerics-package</td>
<td>3</td>
</tr>
<tr>
<td>annotation</td>
<td>6</td>
</tr>
<tr>
<td>aperm</td>
<td>7</td>
</tr>
<tr>
<td>append</td>
<td>8</td>
</tr>
<tr>
<td>as.data.frame</td>
<td>9</td>
</tr>
<tr>
<td>as.list</td>
<td>10</td>
</tr>
<tr>
<td>as.vector</td>
<td>11</td>
</tr>
<tr>
<td>boxplot</td>
<td>12</td>
</tr>
<tr>
<td>cbind</td>
<td>13</td>
</tr>
<tr>
<td>combine</td>
<td>14</td>
</tr>
<tr>
<td>dbconn</td>
<td>16</td>
</tr>
<tr>
<td>density</td>
<td>17</td>
</tr>
<tr>
<td>dge</td>
<td>18</td>
</tr>
<tr>
<td>dims</td>
<td>19</td>
</tr>
<tr>
<td>do.call</td>
<td>20</td>
</tr>
<tr>
<td>duplicated</td>
<td>21</td>
</tr>
<tr>
<td>eval</td>
<td>23</td>
</tr>
<tr>
<td>evalq</td>
<td>24</td>
</tr>
<tr>
<td>Extremes</td>
<td>24</td>
</tr>
<tr>
<td>fileName</td>
<td>26</td>
</tr>
<tr>
<td>format</td>
<td>26</td>
</tr>
<tr>
<td>funprog</td>
<td>27</td>
</tr>
<tr>
<td>get</td>
<td>29</td>
</tr>
<tr>
<td>grep</td>
<td>30</td>
</tr>
<tr>
<td>image</td>
<td>31</td>
</tr>
<tr>
<td>IQR</td>
<td>32</td>
</tr>
<tr>
<td>is.unsorted</td>
<td>33</td>
</tr>
<tr>
<td>lapply</td>
<td>34</td>
</tr>
<tr>
<td>mad</td>
<td>35</td>
</tr>
<tr>
<td>mapply</td>
<td>36</td>
</tr>
<tr>
<td>match</td>
<td>37</td>
</tr>
<tr>
<td>mean</td>
<td>38</td>
</tr>
<tr>
<td>normalize</td>
<td>39</td>
</tr>
<tr>
<td>nrow</td>
<td>40</td>
</tr>
<tr>
<td>Ontology</td>
<td>41</td>
</tr>
<tr>
<td>order</td>
<td>42</td>
</tr>
<tr>
<td>organism_species</td>
<td>43</td>
</tr>
<tr>
<td>paste</td>
<td>45</td>
</tr>
<tr>
<td>path</td>
<td>46</td>
</tr>
<tr>
<td>plotMA</td>
<td>48</td>
</tr>
<tr>
<td>plotPCA</td>
<td>49</td>
</tr>
</tbody>
</table>
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::.aperm
• BiocGenerics::append
  • BiocGenerics::as.data.frame
  • BiocGenerics::as.list
  • BiocGenerics::as.vector
  • BiocGenerics::rbind, BiocGenerics::cbind
  • BiocGenerics::do.call
  • BiocGenerics::duplicated, BiocGenerics::anyDuplicated
  • BiocGenerics::eval
  • Extremes: BiocGenerics::pmax, BiocGenerics::pmin, BiocGenerics::pmax.int, BiocGenerics::pmin.int
  • BiocGenerics::format
  • 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::t
  • BiocGenerics::table
  • BiocGenerics::tapply
  • BiocGenerics::unique
  • BiocGenerics::unlist
  • BiocGenerics::unsplit
  • BiocGenerics::which
  • BiocGenerics::which.min, BiocGenerics::which.max

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

(2) Bioconductor specific generics:
- annotation, annotation<-
- combine
- dbconn, dbfile
- counts, counts<-, design, design<-, dispTable, dispTable<-, sizeFactors, sizeFactors<-, conditions, conditions<-, estimateSizeFactors, estimateDispersions, plotDispEsts
- dims, nrows, ncols,
- 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 Biocon-devel mailing list:

http://bioconductor.org/help/mailing-list/

Things that should NOT be added to the BiocGenerics package:

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

**Author(s)**

The Bioconductor Dev Team
See Also

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

Examples

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

---

**annotation**

Accessing annotation information

Description

Get or set the annotation information contained in an object.

Usage

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

Arguments

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

See Also

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

Examples

```r
annotation
showMethods("annotation")

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

Transposing an array-like object

Description

Transpose an array-like object by permuting its dimensions.

This is a multidimensional generalization of the \texttt{t()} operator used for 2D-transposition.

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

Usage

\texttt{aperm(a, perm, \ldots)}

Arguments

\begin{itemize}
  \item \texttt{a} \hspace{1cm} An array-like object.
  \item \texttt{perm, \ldots} \hspace{1cm} See \texttt{?base::aperm} for a description of these arguments.
\end{itemize}

Value

A transposed version of array-like object \texttt{a}, with subscripts permuted as indicated by the \texttt{perm} vector.

See Also

\begin{itemize}
  \item \texttt{base::aperm} for the default \texttt{aperm} method.
  \item \texttt{showMethods} for displaying a summary of the methods defined for a given generic function.
  \item \texttt{selectMethod} for getting the definition of a specific method.
  \item \texttt{aperm.DelayedArray-method} in the \textbf{DelayedArray} package for an example of a specific \texttt{aperm} method (defined for \texttt{DelayedArray} objects).
  \item \textbf{BiocGenerics} for a summary of all the generics defined in the \textbf{BiocGenerics} package.
\end{itemize}

Examples

\begin{verbatim}
aperm   # note the dispatch on the 'a' arg only
showMethods("aperm")
selectMethod("aperm", "ANY")  # the default method
\end{verbatim}
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

Coerce to a data frame

Description

Generic 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

An ordinary 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.list  

Coerce to a list

Description

Generic function to coerce to a list, if possible.

NOTE: This man page is for the as.list S4 generic function defined in the BiocGenerics package. See ?base::as.list 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.list(x, ...)

Arguments

x 

The object to coerce.

... 

Additional arguments, for use in specific methods.

Value

An ordinary list.

See Also

- base::as.list for the default as.list 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.list.List-method in the S4Vectors package for an example of a specific as.list method (defined for List objects).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.

Examples

as.list
showMethods("as.list")
selectMethod("as.list", "ANY")  # the default method

library(S4Vectors)
showMethods("as.list")
## The as.list() method for List objects:
selectMethod("as.list", "List")
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 ?base::as.vector for a description of this argument.

Value

A vector.

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

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

See Also

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

Examples

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

library(affy)
showMethods("boxplot")
## The boxplot() method for AffyBatch objects:
selectMethod("boxplot", "AffyBatch")
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

```r
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,RectangularData-method` and `cbind,DataFrame-method` in the `S4Vectors` package for examples of specific `rbind` and `cbind` methods (defined for `RectangularData` derivatives and `DataFrame` objects, respectively).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

```r
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

library(S4Vectors)
showMethods("rbind")
## The rbind() method for RectangularData derivatives:
selectMethod("rbind", "RectangularData")
## The cbind() method for DataFrame objects:
selectMethod("cbind", "DataFrame")

---

**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**

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

**Details**

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

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

**Value**

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

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

combine(x=ANY, missing) Return the first (x) argument unchanged.

combine(data.frame, data.frame) Combines two data.frame objects so that the resulting data.frame contains all rows and columns of the original objects. Rows and columns in the returned value are unique, that is, a row or column represented in both arguments is represented only once in the result. To perform this operation, combine makes sure that data in shared rows and columns are identical in the two 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

- 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

```r
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.
density

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

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

Arguments

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

Get the dimensions of each element of a list-like object

**Description**

Get the dimensions, number of rows, or number of columns, of each element of a list-like object.

Note that these functions are the vectorized versions of corresponding functions `dim()`, `nrow()`, and `ncol()`, in the same fashion that `lengths()` is the vectorized version of `length`.

**Usage**

```r
dims(x, use.names=TRUE)
nrows(x, use.names=TRUE)
ncols(x, use.names=TRUE)
```

**Arguments**

- `x` List-like object (or environment) where all the list elements are expected to be array-like objects with the same number of dimensions.
- `use.names` Logical indicating if the names on `x` should be propagated to the returned matrix (as its rownames) or vector (as its names).

**Value**

For `dims()`: Typically a numeric matrix with one row per list element in `x` and one column per dimension in these list elements (they’re all expected to have the same number of dimensions). The i-th row in the returned matrix is a vector containing the dimensions of the i-th list element in `x`. More formally:

`dims(x)[i, ]` is `dim(x[[i]])`
for any valid i. By default the names on x, if any, are propagated as the rownames of the returned matrix, unless use.names is set to FALSE.

For nrows() or ncols(): A numeric vector with one element per list element in x. The i-th element in the returned vector is the number of rows (or columns) of the i-th list element in x. More formally:

\[
nrows(x)[i] \text{ is } nrow(x[[i]]) \text{ and } ncols(x)[i] \text{ is } ncol(x[[i]])
\]

for any valid i. By default the names on x, if any, are propagated on the returned vector, unless use.names is set to FALSE.

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

Examples

dims
showMethods("dims")

library(IRanges)
showMethods("dims")
selectMethod("dims", "DataFrameList")

---

**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**

```r
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, ...)
duplicated(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 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

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

Arguments

- **expr**: An object to be evaluated. May be any object supported by the default method (see ?base::eval) or by the additional methods defined in Bioconductor packages.
- **envir**: The environment in which expr is to be evaluated. May be any object supported by the default method (see ?base::eval) or by the additional methods defined in Bioconductor packages.
- **enclos**: See ?base::eval for a description of this argument.

Value

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

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

See Also

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

Examples

```r
eval  # note the dispatch on 'expr' and 'envir' args only
showMethods("eval")
selectMethod("eval", c("ANY", "ANY"))  # the default method
```
evalq \hspace{1em} 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

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

Value

See `?base::evalq`.

See Also

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

Examples

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

Extremes \hspace{1em} Maxima and minima

Description

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

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

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

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

Arguments

... One or more vector-like or matrix-like objects.

na.rm See ?base::pmax for a description of this argument.

Value

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

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

See Also

- base::pmax for the default pmax, pmin, pmax.int and pmin.int methods.
- showMethods for displaying a summary of the methods defined for a given generic function.
- selectMethod for getting the definition of a specific method.
- pmax.Rle-method in the 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

**Accessing the file name of an object**

**Description**

Get the file name of an object.

**Usage**

```r
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**

```r
fileName
showMethods("fileName")

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

---

**format**

**Format an R object for pretty printing**

**Description**

Turn an R object into a character vector used for pretty printing.

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

Usage

format(x, ...)

Arguments

x The object to format.
...

Additional arguments, for use in specific methods.

Value

A character vector that provides a "compact representation" of x. This character vector is typically used by print.data.frame to display the columns of a data.frame object. See ?base::print.data.frame for more information.

See Also

- base::format for the default format 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

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

---

funprog Common higher-order functions in functional programming languages

Description

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

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

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

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

See Also
• base::Reduce for the default Reduce, Filter, Find, Map and Position methods.
• showMethods for displaying a summary of the methods defined for a given generic function.
• selectMethod for getting the definition of a specific method.
• Reduce,List-method in the 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.
grep

Pattern Matching and Replacement

Description

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

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

Usage

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

Arguments

pattern The pattern for searching in x, such as a regular expression.
x The character vector (in the general sense) to search.
ignore.case, perl, value, fixed, useBytes, invert

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

Value

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

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

See Also

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

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

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.AffyBatch-method in the affy package for an example of a specific image method (defined for AffyBatch objects).
- BiocGenerics for a summary of all the generics defined in the BiocGenerics package.
Examples

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

library(affy)
showMethods("image")
## The image() method for AffyBatch objects:
selectMethod("image", "AffyBatch")
```

---

**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**

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

**Arguments**

- `x`: See `?stats::IQR`.
- `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**

```r
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
x  A vector-like object.
na.rm, strictly  See ?base::is.unsorted for a description of these arguments.
...  Additional arguments, for use in specific methods.

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

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

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

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

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

lapply

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

Description

`lapply` returns a list of the same length as `X`, each element of which is the result of applying `FUN` to the corresponding element of `X`.

`sapply` is a user-friendly version and wrapper of `lapply` by default returning a vector, matrix or, if `simplify="array"`, an array if appropriate, by applying `simplify2array()`. `sapply(x, f, simplify=FALSE, USE.NAMES=FALSE)` 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

```r
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.
mad

Examples

```r
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

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

Arguments

- `x`: Numeric vector.
- `center`: An optional function to compute the center of the vector.
- `constant`: A numeric constant.
- `na.rm`: A logical value indicating whether `NA` values should be stripped before the computation proceeds.
- `low`: A logical value indicating whether lower bounds should be used.
- `high`: A logical value indicating whether upper bounds should be used.

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

```r
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

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>match returns a vector of the positions of (first) matches of its first argument in its second.</td>
</tr>
<tr>
<td>%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.</td>
</tr>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

**Usage**

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

```
x %in% table
```

<table>
<thead>
<tr>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, table</td>
</tr>
<tr>
<td>nomatch, incomparables</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

**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

```r
mean # 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

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**

```r
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**

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

Normalize an object

Description

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

Usage

normalize(object, ...)

Arguments

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.MSnExp-method in the MSnbase package for examples of specific normalize methods (defined for AffyBatch and MSnExp 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")
The number of rows/columns of an array-like object

Description

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

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

Usage

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

Arguments

x
A matrix- or array-like object.

Value

A single integer or NULL.

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

See Also

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

Examples

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

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

Generic Ontology getter

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

Ordering permutation

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.
organism_species

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

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

---

organism_species  
Organism and species accessors

Description

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

Usage

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

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

Arguments

- object An object to get or set the organism or species of.
- value The organism or species to set on object.

Value

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

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

TO DEVELOPERS:

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

**See Also**

- [http://bioconductor.org/packages/release/BiocViews.html#___Organism](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**

```r
## 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<-")
```
**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**

```r
paste(..., sep=" ", collapse=NULL, recycle0=FALSE)
```

**Arguments**

- `...` One or more vector-like objects containing strings.
- `sep`, `collapse`, `recycle0`
  
  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).

**Examples**

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

Get or set the path of an object.

Usage

```r
path(object, ...)     # Get the path of an object.
object <- path(object, ...)  # Set the path of an object.

basename(path, ...)     # Get the base name of a path.
basename(path, ...) <- value  # Set the base name of a path.

dirname(path, ...)     # Get the directory of a path.
dirname(path, ...) <- value  # Set the directory of a path.
```

Arguments

- `object` An object containing paths. Even though it will typically contain a single path, `object` can actually contain an arbitrary number of paths.
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 or an object containing paths.

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

```r
## 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")
```
## OBJECTS CONTAINING PATHS

Let’s define a simple class to represent objects that contain paths:

```r
setClass("A", slots=c(stuff="ANY", path="character"))
```

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

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

```r
path(a)
```

### Because the path() getter works on ‘a’, now the basename() and
### dirname() getters also work:
```r
basename(a)
dirname(a)
```

### path() setter:
```r
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
   })
```

```r
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:
```r
basename(a) <- toupper(basename(a))
path(a)
dirname(a) <- "/MyDataFiles"
path(a)
```
**plotPCA**

PCA-plot: Principal Component Analysis plot

**Description**

A generic function which produces a PCA-plot.

**Usage**

plotPCA(object, ...)

**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)
)

---

---
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

```r
showMethods("plotPCA")

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

Description

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

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

Usage

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

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

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

Re-listing an unlist()ed object

Description

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

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

Usage

relist(flesh, skeleton)

Arguments

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

Value

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

See Also

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

Examples

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

x  The object to replicate (typically vector-like).

times See ?base::rep.int for a description of this argument.

Value

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

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

See Also

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

Examples

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

Description

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

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

Usage

residuals(object, ...)

Arguments

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

Value

Residuals extracted from the object object.

See Also

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

Examples

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

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

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

Arguments

- **x**  
  A matrix-like object.

- **do.NULL**  
  See `?base::rownames` for a description of these arguments.

- **prefix**  
  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

```r
## 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 BiocGenerics package. This allows S4 methods defined in Bioconductor packages to use them in their signatures.

**Details**

S3 classes currently turned into S4 classes:

- connection class and subclasses: connection, file, url, gzfile, bzfile, unz, pipe, fifo, sockconn, terminal, textConnection, gzcon. Additionally the character.OR.connection 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

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.
...

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 \texttt{?base::union} for the default methods (defined in the \texttt{base} package). Bioconductor packages can define specific methods for objects (typically vector-like) not supported by the default methods.

Usage

\begin{verbatim}
union(x, y, ...)  
intersect(x, y, ...)
setdiff(x, y, ...)
\end{verbatim}

Arguments

\begin{verbatim}
x, y  Vector-like objects (typically of the same class, but not necessarily).
...  Additional arguments, for use in specific methods.
\end{verbatim}

Value

See \texttt{?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 \texttt{base} package) only take 2 arguments. We’ve added the \ldots argument to the generic functions defined in the BiocGenerics package so they can be called with an arbitrary number of effective arguments. For \texttt{union} or \texttt{intersect}, this typically allows Bioconductor packages to define methods that compute the union or intersection of more than 2 objects. However, for \texttt{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 \texttt{ignore.strand} argument supported by the \texttt{setdiff} method for GenomicRanges objects (defined in the GenomicRanges package). (Note that the \texttt{union} and \texttt{intersect} methods for those objects also support the \texttt{ignore.strand} argument.)

See Also

\begin{itemize}
  \item \texttt{base::union} for the default \texttt{union}, \texttt{intersect}, and \texttt{setdiff} methods.
  \item \texttt{showMethods} for displaying a summary of the methods defined for a given generic function.
  \item \texttt{selectMethod} for getting the definition of a specific method.
\end{itemize}
sort


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

Examples

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

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

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

sort

**Sorting a vector-like object**

Description

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

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

Usage

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

Arguments

- `x` A vector-like object.

- `decreasing,...` See ?base::sort for a description of these arguments.

Value

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

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

TO DEVELOPERS:

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

\code{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 \code{<=} method.

See Also

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

Examples

\begin{verbatim}
sort  # note the dispatch on the 'x' arg only
showMethods("sort")
selectMethod("sort", "ANY")  # the default method
\end{verbatim}

\--

\抱

\section*{Description}

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

NOTE: This man page is for the \code{start}, \code{end}, \code{width}, \code{width<->}, and \code{pos} \Sexpr{S4 generic functions} defined in the \pkg{BiocGenerics} package. See \code{?stats::start} for the \code{start} and \code{end} S3 generics defined in the \pkg{stats} package.

Usage

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

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

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

pos(x)
\end{verbatim}

Arguments

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

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:
strand  

Accessing strand information

Description

Get or set the strand information contained in an object.

Usage

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

unstrand(x)  

invertStrand(x)  

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

Arguments

x  
An object containing strand information.

...  
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:
The default method for `invertStrand` does:

```r
strand(x) <- invertStrand(strand(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**

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

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

library(GenomicRanges)
showMethods("strand")
selectMethod("strand", "missing")
strand()
showMethods("strand<-")
```
Description

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

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

Usage

```r
subset(x, ...)  
```

Arguments

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

Value

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

See Also

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

Examples

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

library(S4Vectors)
showMethods("subset")

## The subset() method for RectangularData derivatives:
sellectionMethod("subset", "RectangularData")
```
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 \textbf{BiocGenerics} package. See \?base::t for the default method (defined in the \textbf{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 matrix-like or other rectangular object.

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

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

Examples

\[
t
\]
\[
\text{showMethods("t")}
\]
\[
\text{selectMethod("t", "ANY") \# the default method}
\]
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.
testPackage

Examples

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

testPackage  Run RUnit package unit tests

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.
toTable

See Also

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

Examples

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

## End(Not run)
```

---

toTable  

An alternative to as.data.frame()

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”)

Description

Accessing the type of an object

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 the type and/or type<- methods are defined.

Usage

type(x)
type(x) <- value

## Methods defined in the BiocGenerics package:

## S4 method for signature 'vector'
type(x)
## S4 method for signature 'array'
type(x)
## S4 method for signature 'factor'
type(x) # returns "character"
## S4 method for signature 'data.frame'
type(x)

## S4 replacement method for signature 'vector'
type(x) <- value
## S4 replacement method for signature 'array'
type(x) <- value

Arguments

x Any object for which the type() 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 both the getter and setter.

value The type to set on x (assuming x supports the type() setter). value is typically (but not necessarily) expected to be a single string (i.e. a character vector of length 1).
Details

On an ordinary vector, matrix, or array \( x \), \( \text{type}(x) \) returns \( \text{typeof}(x) \).

On a data frame \( x \) where all the columns are ordinary vectors or factors, \( \text{type}(x) \) is \textit{semantically equivalent} to \( \text{typeof}(\text{as.matrix}(x)) \). However, the actual implementation is careful to avoid turning the full data frame \( x \) into a matrix, as this would tend to be very inefficient in general.

Note that for a matrix-like or array-like object, \( \text{type}(x) \) returns the type of the \textit{elements} in the object. See \(?S4Arrays::type\) for more information.

Value

\( \text{type}(x) \) is expected to return the type of \( x \) as a single string i.e. as a character vector of length 1.

See Also

- \texttt{showMethods} for displaying a summary of the methods defined for a given generic function.
- \texttt{selectMethod} for getting the definition of a specific method.
- \texttt{type,ANY-method} in the \texttt{DelayedArray} package for the default \texttt{type} method.
- \texttt{type,DataFrame-method} in the \texttt{S4Arrays} package, and \texttt{type,PairwiseAlignments-method} in the \texttt{Biostrings} package, for examples of specific \texttt{type} methods (defined for \texttt{DataFrame} and \texttt{PairwiseAlignments} objects, respectively).
- \texttt{BiocGenerics} for a summary of all the generics defined in the \texttt{BiocGenerics} package.

Examples

```r
\texttt{type}
\texttt{showMethods("type")}

\texttt{\`type<-\`}
\texttt{showMethods("\`type<-\")}

## The BiocGenerics package defines methods for ordinary vectors, arrays, and data frames:
\texttt{m <- matrix(11:22, nrow=3)}
\texttt{type(m)} # equivalent to '\texttt{typeof(m)}' or '\texttt{storage.mode(m)}'
\texttt{type(m) <- "raw" # equivalent to \texttt{\`storage.mode\`}(m) <- \texttt{"raw"'}}
\texttt{m}
\texttt{type(m)}

\texttt{selectMethod("\`type\`, \"array\")}

\texttt{selectMethod("\`type<-\", \"array\")}

\texttt{df <- data.frame(a=44:49, b=letters[1:6], c=c(TRUE, FALSE))}
\texttt{stopifnot(identical(type(df), typeof(as.matrix(df))))}

## Examples of methods defined in other packages:

\texttt{library(DelayedArray)}
\texttt{showMethods("\`type\")}
```
selectMethod("type", "ANY")  # the default "type" method

library(Biostrings)
showMethods("type")
## The type() method for PairwiseAlignments objects:
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

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

### Arguments

- **x**  
  A vector-like, data-frame-like, or array-like object.

- **incomparables,**  
  See ?base::unique for a description of these arguments.

### Value

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

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

`unique` should always behave consistently with BiocGenerics::duplicated.

### See Also

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

Examples

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

unlist

Flatten list-like objects

Description

Given a list-like object \( x \), \texttt{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 \texttt{unlist} \textit{S4 generic function} defined in the \texttt{BiocGenerics} package. See `?base::unlist` for the default method (defined in the \texttt{base} package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

\texttt{unlist(x, recursive=TRUE, use.names=TRUE)}

Arguments

- \texttt{x} A list-like object.
- \texttt{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 \texttt{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 \texttt{S4Vectors} package for an example of a specific \texttt{unlist} method (defined for \texttt{List} objects).
- `BiocGenerics` for a summary of all the generics defined in the \texttt{BiocGenerics} package.

Examples

\texttt{unlist}  # note the dispatch on the 'x' arg only
showMethods("unlist")
selectMethod("unlist", "ANY")  # the default method
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 BioGenerics package. See ?base::unsplit for the default method (defined in the base package). Bioconductor packages can define specific methods for objects not supported by the default method.

Usage

unsplit(value, f, drop=FALSE)

Arguments

value       A list-like object.
f           A factor or other grouping object that corresponds to the f symbol in value <- split(x, f).
drop        See ?base::unsplit for a description of this argument.

Value

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

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

See Also

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

Examples

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

Update an object to its current class definition

**Description**

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

**Usage**

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

## Related utilities:

- `updateObjectFromSlots(object, objclass=class(object)[[1L]], ..., verbose=FALSE)`
- `getObjectSlots(object)`

**Arguments**

- `object` Object to be updated for `updateObject` and `updateObjectFromSlots`.
- `...` 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:

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

---

**updateObjectFromSlots**

Update an object to its current class definition

**Description**

`updateObjectFromSlots` is a function that returns an object updated to its current class definition.

**Usage**

```r
updateObjectFromSlots(object, objclass=class(object)[[1L]], ..., verbose=FALSE)
```

**Arguments**

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

**Details**

`updateObjectFromSlots` 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 `validObjectFromSlots`). By default, `updateObjectFromSlots` has the following behaviors:

- `updateObjectFromSlotsANY, ..., verbose=FALSE)` By default, `updateObjectFromSlots` 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`.
- `updateObjectFromSlots(list, ..., verbose=verbose)` Visit each element in `list`, applying `updateObjectFromSlots(list[[elt]], ..., verbose=verbose)`.
- `updateObjectFromSlots(environment, ..., verbose=verbose)` Visit each element in `environment`, applying `updateObjectFromSlots(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=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.

ggetObjectSlots(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.
ggetObjectSlots 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'")
  

var <- callNextMethod()
object@x <- -object@x
object
}

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

---

Variances and Standard Deviations

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

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

Arguments

- `x`: a vector-like object
- `y`: a vector-like object, or NULL
- `na.rm`, `use`: 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.
weights

Extract model weights

Description

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

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

Usage

```r
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

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

Give the indices of the values in a vector-, array-, or list-like object that are considered TRUE, allowing for array indices in the case of an array-like object.

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

Usage

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

Arguments

x

An object, typically with a vector-, array-, or list-like semantic.

arr.ind, useNames

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

Value

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

Examples

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

library(DelayedArray)
showMethods("which")
## The which() method for DelayedArray objects:
selectMethod("which", "DelayedArray")
which.min

What’s the index of the first min or max value in an object?

Description

Determines the location (i.e. index) of the (first) minimum or maximum value in an object.

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

Usage

```r
which.min(x, ...)
which.max(x, ...)
```

Arguments

- `x`  
  An object, typically with a vector-, array-, or list-like semantic.

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

Value

See `?base::which.min` 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 methods (defined in the `base` package) only take a single argument. 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.min` for the default `which.min` and `which.max` methods.
- `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.
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.

Examples

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

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

library(IRanges)
showMethods("which.max")
## The which.max() method for NumericList objects:
selectMethod("which.max", "NumericList")
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,DataFrame-method` in the `S4Vectors` package for an example of a specific `xtabs` method (defined for `DataFrame` objects).
- `BiocGenerics` for a summary of all the generics defined in the `BiocGenerics` package.

Examples

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

library(S4Vectors)
showMethods("xtabs")
## The xtabs() method for DataFrame objects:
selectMethod("xtabs", "DataFrame")
```
Index

* classes
  S3-classes-as-S4-classes, 56
* manip
dge, 18
* methods
  annotation, 6
  aperm, 7
  append, 8
  as.data.frame, 9
  as.list, 10
  as.vector, 11
  boxplot, 12
  cbind, 13
  combine, 14
  dbconn, 16
  density, 17
  dims, 19
  do.call, 20
  duplicated, 21
  eval, 23
  Extremes, 24
  fileName, 26
  format, 26
  funprog, 27
  get, 29
  grep, 30
  image, 31
  IQR, 32
  is.unsorted, 33
  lapply, 34
  mad, 35
  mapply, 36
  match, 37
  mean, 38
  normalize, 39
  nrow, 40
  Ontology, 41
  order, 42
  organism_species, 43
  paste, 45
  path, 46
  plotMA, 48
  plotPCA, 49
  rank, 50
  relist, 52
  rep, 53
  residuals, 54
  row+colnames, 55
  score, 57
  sets, 58
  sort, 59
  start, 60
  strand, 62
  subset, 64
  t, 65
  table, 66
  tapply, 67
  tapply, 67
  testPackage, 68
  toTable, 69
  type, 70
  unique, 72
  unlist, 73
  unsplit, 74
  updateObject, 75
  var, 77
  weights, 78
  which, 79
  which.min, 80
  xtabs, 81
* package
  BiocGenerics-package, 3
  <=, 42, 51, 60
  %in%(match), 37
  %in%, 4
  %in%,Rle,ANY-method, 37

  AffyBatch, 12, 31, 39
  AnnotatedDataFrame, 15
  annotation, 5, 6
<table>
<thead>
<tr>
<th>Function</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotation.eSet-method</td>
<td>6</td>
</tr>
<tr>
<td>annotation&lt;- (annotation)</td>
<td>6</td>
</tr>
<tr>
<td>AnnotationDb</td>
<td>44</td>
</tr>
<tr>
<td>anyDuplicated</td>
<td>4</td>
</tr>
<tr>
<td>anyDuplicated (duplicated)</td>
<td>21</td>
</tr>
<tr>
<td>aperm</td>
<td>3, 7, 7</td>
</tr>
<tr>
<td>aperm.DelayedArray-method</td>
<td>7</td>
</tr>
<tr>
<td>append, Vector, Vector-method</td>
<td>8</td>
</tr>
<tr>
<td>as.data.frame</td>
<td>4, 9, 69</td>
</tr>
<tr>
<td>as.data.frame.DataFrame-method</td>
<td>9</td>
</tr>
<tr>
<td>as.data.frame, IntegerRanges-method</td>
<td>9</td>
</tr>
<tr>
<td>as.list</td>
<td>4, 10, 10</td>
</tr>
<tr>
<td>as.list.List-method</td>
<td>10</td>
</tr>
<tr>
<td>as.vector</td>
<td>4, 11, 11</td>
</tr>
<tr>
<td>as.vector, AtomicList-method</td>
<td>11</td>
</tr>
<tr>
<td>as.vector, Rle-method</td>
<td>11</td>
</tr>
<tr>
<td>AsIs</td>
<td>56</td>
</tr>
<tr>
<td>AsIs-class (S3-classes-as-S4-classes)</td>
<td>56</td>
</tr>
<tr>
<td>AssayData</td>
<td>15</td>
</tr>
<tr>
<td>AtomicList</td>
<td>11</td>
</tr>
<tr>
<td>basename</td>
<td>5, 47</td>
</tr>
<tr>
<td>basename (path)</td>
<td>46</td>
</tr>
<tr>
<td>basename, ANY-method (path)</td>
<td>46</td>
</tr>
<tr>
<td>basename&lt;- (path)</td>
<td>46</td>
</tr>
<tr>
<td>basename&lt;-, ANY-method (path)</td>
<td>46</td>
</tr>
<tr>
<td>basename&lt;-, character-method (path)</td>
<td>46</td>
</tr>
<tr>
<td>Bimap</td>
<td>29, 69</td>
</tr>
<tr>
<td>BiocGenerics (BiocGenerics-package)</td>
<td>3</td>
</tr>
<tr>
<td>BiocGenerics-package</td>
<td>3</td>
</tr>
<tr>
<td>boxplot</td>
<td>4, 12, 12</td>
</tr>
<tr>
<td>boxplot, AffyBatch-method</td>
<td>12</td>
</tr>
<tr>
<td>bzfile-class (S3-classes-as-S4-classes)</td>
<td>56</td>
</tr>
<tr>
<td>c. 73</td>
<td></td>
</tr>
<tr>
<td>cbind</td>
<td>4, 5, 13, 13</td>
</tr>
<tr>
<td>cbind, DataFrame-method</td>
<td>13</td>
</tr>
<tr>
<td>character.OR.connection-class (S3-classes-as-S4-classes)</td>
<td>56</td>
</tr>
<tr>
<td>chromLocation</td>
<td>44</td>
</tr>
<tr>
<td>colnames</td>
<td>4</td>
</tr>
<tr>
<td>colnames (row+colnames)</td>
<td>55</td>
</tr>
<tr>
<td>colnames&lt;- (row+colnames)</td>
<td>55</td>
</tr>
<tr>
<td>combine</td>
<td>5, 14</td>
</tr>
<tr>
<td>combine, AnnotatedDataFrame, AnnotatedDataFrame-method</td>
<td>15</td>
</tr>
<tr>
<td>combine, ANY, missing-method (combine)</td>
<td>14</td>
</tr>
<tr>
<td>combine, AssayData, AssayData-method</td>
<td>15</td>
</tr>
<tr>
<td>combine, data.frame, data.frame-method (combine)</td>
<td>14</td>
</tr>
<tr>
<td>combine, eSet, eSet-method</td>
<td>15</td>
</tr>
<tr>
<td>combine, matrix, matrix-method (combine)</td>
<td>14</td>
</tr>
<tr>
<td>combine, MIAME, MIAME-method</td>
<td>15</td>
</tr>
<tr>
<td>conditions</td>
<td>5</td>
</tr>
<tr>
<td>conditions (dge)</td>
<td>18</td>
</tr>
<tr>
<td>conditions&lt;- (dge)</td>
<td>18</td>
</tr>
<tr>
<td>connection</td>
<td>56</td>
</tr>
<tr>
<td>connection-class (S3-classes-as-S4-classes)</td>
<td>56</td>
</tr>
<tr>
<td>counts</td>
<td>5</td>
</tr>
<tr>
<td>counts (dge)</td>
<td>18</td>
</tr>
<tr>
<td>counts&lt;- (dge)</td>
<td>18</td>
</tr>
<tr>
<td>DataFrame</td>
<td>9, 13, 40, 55, 71, 82</td>
</tr>
<tr>
<td>DataFrameList</td>
<td>20</td>
</tr>
<tr>
<td>dbconn</td>
<td>5, 16, 17</td>
</tr>
<tr>
<td>dbconn, AnnotationDb-method</td>
<td>17</td>
</tr>
<tr>
<td>dbfile</td>
<td>5</td>
</tr>
<tr>
<td>dbfile (dbconn)</td>
<td>16</td>
</tr>
<tr>
<td>DelayedArray</td>
<td>7, 79</td>
</tr>
<tr>
<td>density</td>
<td>5, 17, 17, 18</td>
</tr>
<tr>
<td>density, flowClust-method</td>
<td>18</td>
</tr>
<tr>
<td>design</td>
<td>5</td>
</tr>
<tr>
<td>design (dge)</td>
<td>18</td>
</tr>
<tr>
<td>design&lt;- (dge)</td>
<td>18</td>
</tr>
<tr>
<td>dge</td>
<td>18</td>
</tr>
<tr>
<td>dim</td>
<td>5</td>
</tr>
<tr>
<td>dims</td>
<td>5, 19</td>
</tr>
<tr>
<td>dims, DataFrameList-method</td>
<td>20</td>
</tr>
<tr>
<td>dirname</td>
<td>5</td>
</tr>
<tr>
<td>dirname (path)</td>
<td>46</td>
</tr>
<tr>
<td>dirname, ANY-method (path)</td>
<td>46</td>
</tr>
<tr>
<td>dirname&lt;- (path)</td>
<td>46</td>
</tr>
<tr>
<td>dirname&lt;-, ANY-method (path)</td>
<td>46</td>
</tr>
<tr>
<td>dirname&lt;-, character-method (path)</td>
<td>46</td>
</tr>
<tr>
<td>dispTable</td>
<td>5</td>
</tr>
<tr>
<td>dispTable (dge)</td>
<td>18</td>
</tr>
<tr>
<td>dispTable&lt;- (dge)</td>
<td>18</td>
</tr>
<tr>
<td>dist</td>
<td>56</td>
</tr>
<tr>
<td>dist-class (S3-classes-as-S4-classes)</td>
<td>56</td>
</tr>
<tr>
<td>Function/Method</td>
<td>Page Numbers</td>
</tr>
<tr>
<td>-----------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>do.call</td>
<td>4, 20, 21</td>
</tr>
<tr>
<td>duplicated</td>
<td>4, 21, 22, 22, 72</td>
</tr>
<tr>
<td>duplicated, Rle-method</td>
<td>22</td>
</tr>
<tr>
<td>end</td>
<td>4</td>
</tr>
<tr>
<td>end(start)</td>
<td>60</td>
</tr>
<tr>
<td>end&lt;- (start)</td>
<td>60</td>
</tr>
<tr>
<td>eSet</td>
<td>15</td>
</tr>
<tr>
<td>estimateDispersions</td>
<td>5</td>
</tr>
<tr>
<td>estimateDispersions (dge)</td>
<td>18</td>
</tr>
<tr>
<td>estimateSizeFactors</td>
<td>5</td>
</tr>
<tr>
<td>estimateSizeFactors (dge)</td>
<td>18</td>
</tr>
<tr>
<td>eval, expression, Vector-method</td>
<td>23</td>
</tr>
<tr>
<td>evalq</td>
<td>24</td>
</tr>
<tr>
<td>expression</td>
<td>23</td>
</tr>
<tr>
<td>Extremes</td>
<td>24</td>
</tr>
<tr>
<td>factor</td>
<td>15</td>
</tr>
<tr>
<td>fifo-class (S3-classes-as-S4-classes)</td>
<td>56</td>
</tr>
<tr>
<td>file-class (S3-classes-as-S4-classes)</td>
<td>56</td>
</tr>
<tr>
<td>fileName</td>
<td>5, 26</td>
</tr>
<tr>
<td>fileName, MSmap-method</td>
<td>26</td>
</tr>
<tr>
<td>Filter</td>
<td>4</td>
</tr>
<tr>
<td>Filter (funprog)</td>
<td>27</td>
</tr>
<tr>
<td>Find</td>
<td>4</td>
</tr>
<tr>
<td>Find(funprog)</td>
<td>27</td>
</tr>
<tr>
<td>flowClust</td>
<td>18</td>
</tr>
<tr>
<td>format, expression</td>
<td>23</td>
</tr>
<tr>
<td>funprog</td>
<td>27</td>
</tr>
<tr>
<td>GenomicRanges</td>
<td>33, 57–59</td>
</tr>
<tr>
<td>get, ANY, Bimap, missing-method</td>
<td>29</td>
</tr>
<tr>
<td>getObjectSlots (updateObject)</td>
<td>75</td>
</tr>
<tr>
<td>GOTerms</td>
<td>41</td>
</tr>
<tr>
<td>GRanges</td>
<td>63</td>
</tr>
<tr>
<td>grep, expression</td>
<td>30</td>
</tr>
<tr>
<td>grepl</td>
<td>4</td>
</tr>
<tr>
<td>grepl (grep)</td>
<td>30</td>
</tr>
<tr>
<td>groupGeneric</td>
<td>5</td>
</tr>
<tr>
<td>gzcon-class (S3-classes-as-S4-classes)</td>
<td>56</td>
</tr>
<tr>
<td>gzfile-class</td>
<td>(S3-classes-as-S4-classes), 56</td>
</tr>
<tr>
<td>Hits</td>
<td>37, 65</td>
</tr>
<tr>
<td>Image</td>
<td>4, 31, 31</td>
</tr>
<tr>
<td>Image, AffyBatch-method</td>
<td>31</td>
</tr>
<tr>
<td>IntegerRanges</td>
<td>9, 43</td>
</tr>
<tr>
<td>InternalMethods</td>
<td>5</td>
</tr>
<tr>
<td>intersect</td>
<td>4</td>
</tr>
<tr>
<td>intersect (sets)</td>
<td>58</td>
</tr>
<tr>
<td>invertStrand</td>
<td>5</td>
</tr>
<tr>
<td>invertStrand (strand)</td>
<td>62</td>
</tr>
<tr>
<td>invertStrand, ANY-method (strand)</td>
<td>62</td>
</tr>
<tr>
<td>IQR</td>
<td>32, 32</td>
</tr>
<tr>
<td>IRanges</td>
<td>61</td>
</tr>
<tr>
<td>is.unsorted</td>
<td>4, 33, 33</td>
</tr>
<tr>
<td>is.unsorted, GenomicRanges-method</td>
<td>33</td>
</tr>
<tr>
<td>lapply, List-method</td>
<td>34</td>
</tr>
<tr>
<td>length</td>
<td>5</td>
</tr>
<tr>
<td>List, 10, 28, 34, 52, 73, 74</td>
<td></td>
</tr>
<tr>
<td>mad</td>
<td>35, 35</td>
</tr>
<tr>
<td>Map</td>
<td>4</td>
</tr>
<tr>
<td>Map (funprog)</td>
<td>27</td>
</tr>
<tr>
<td>mapply, 4, 36, 36</td>
<td></td>
</tr>
<tr>
<td>match, 4, 37, 37</td>
<td></td>
</tr>
<tr>
<td>match, Hits, Hits-method</td>
<td>37</td>
</tr>
<tr>
<td>Math, 5</td>
<td></td>
</tr>
<tr>
<td>mean, 38, 38</td>
<td></td>
</tr>
<tr>
<td>mean, Rle-method</td>
<td>38</td>
</tr>
<tr>
<td>merge, 15</td>
<td></td>
</tr>
<tr>
<td>message</td>
<td>75</td>
</tr>
<tr>
<td>mget, 4</td>
<td></td>
</tr>
<tr>
<td>mget (get)</td>
<td>29</td>
</tr>
<tr>
<td>MIAME</td>
<td>15</td>
</tr>
<tr>
<td>MSmap</td>
<td>26</td>
</tr>
<tr>
<td>MSnExp</td>
<td>39</td>
</tr>
<tr>
<td>NCOL, 4</td>
<td></td>
</tr>
<tr>
<td>NCOL (nrow)</td>
<td>40</td>
</tr>
<tr>
<td>ncol, 4, 55</td>
<td></td>
</tr>
<tr>
<td>ncol (nrow)</td>
<td>40</td>
</tr>
<tr>
<td>ncol2s, 5</td>
<td></td>
</tr>
<tr>
<td>ncol2s (dims)</td>
<td>19</td>
</tr>
<tr>
<td>normalize, 5, 39</td>
<td></td>
</tr>
<tr>
<td>normalize, AffyBatch-method</td>
<td>39</td>
</tr>
<tr>
<td>normalize, MSnExp-method</td>
<td>39</td>
</tr>
<tr>
<td>NROW, 4</td>
<td></td>
</tr>
<tr>
<td>NROW (nrow)</td>
<td>40</td>
</tr>
<tr>
<td>nrow, 4, 40, 40, 55</td>
<td></td>
</tr>
<tr>
<td>nrow, DataFrame-method</td>
<td>40</td>
</tr>
</tbody>
</table>
nrows, 5
nrows (dims), 19
NumericList, 80

Ontology, 5, 41
Ontology, GOTerms-method, 41
Ops, 5
order, 4, 33, 42, 43, 51, 60
order, IntegerRanges-method, 43
organism, 5
organism (organism_species), 43
organism, character-method, 44
organism, chromLocation-method, 44
organism<- (organism_species), 43
organism_species, 43

PairwiseAlignments, 71
paste, 4, 45, 45
paste, Rle-method, 45
path, 5, 46
path, RsamtoolsFile-method, 47
path<- (path), 46
pipe-class (S3-classes-as-S4-classes), 56
PLMset, 54, 78
plotDispEsts, 5
plotDispEsts (dge), 18
plotMA, 5, 48, 49
plotMA, ANY-method (plotMA), 48
plotPCA, 5, 49, 50
pmax, 4, 24, 25
pmax (Extremes), 24
pmax, Rle-method, 25
pmax.int, 4
pmin, 4
pmin (Extremes), 24
pmin.int, 4
pos, 4
pos (start), 60
pos, UnstitchedIPos-method, 61
Position, 4
Position (funprog), 27
print.data.frame, 27

rank, 4, 33, 42, 50, 51, 60
rank, Vector-method, 51
rbind, 4
rbind (cbind), 13
rbind, RectangularData-method, 13
RectangularData, 13, 64
Reduce, 4, 27, 28
Reduce (funprog), 27
Reduce, List-method, 28
relist, 4, 52, 52
relist, ANY, List-method, 52
rep, 53
rep.int, 4, 53
relist, Rle-method, 53
residuals, 5, 54, 54
residuals, PLMset-method, 54
Rle, 11, 22, 25, 37, 38, 43, 53, 56, 72
row+colnames, 55
rownames, 4, 55
rownames (row+colnames), 55
rownames, DataFrame-method, 55
rownames<- (row+colnames), 55
RsamtoolsFile, 47

S3-classes-as-S4-classes, 56
S4groupGeneric, 5
sapply, 4, 36
sapply (lapply), 34
score, 5, 57
score, GenomicRanges-method, 57
score<- (score), 57
sd, 77
sd (var), 77
setClassUnion, 56
setdiff, 4
setdiff (sets), 58
setGeneric, 6
setMethod, 6
setOldClass, 56
sets, 58
sizeFactors, 5
sizeFactors (dge), 18
sizeFactors<- (dge), 18
sockconn-class

(S3-classes-as-S4-classes), 56
sort, 4, 33, 42, 51, 59, 59, 60
sort, Vector-method, 60
species, 5
species (organism_species), 43
species, AnnotationDb-method, 44
species <- (organism_species), 43
start, 4, 60, 60, 61
start <- (start), 60
strand, 5, 62
strand <- (strand), 62
subset, 4, 64, 64
subset, RectangularData-method, 64
t, 4, 7, 65, 65
t, Hits-method, 65
table, 4, 66, 66
table, Rle-method, 66
tapply, 4, 67, 67
tapply, Vector, ANY-method, 67
terminal-class
(S3-classes-as-S4-classes), 56
testPackage, 68
textConnection-class
(S3-classes-as-S4-classes), 56
toTable, 5, 9, 69
toTable, Bimap-method, 69
type, 5, 70, 71
type, ANY-method, 71
type, array-method (type), 70
type, data.frame-method (type), 70
type, DataFrame-method, 71
type, factor-method (type), 70
type, PairwiseAlignments-method, 71
type, vector-method (type), 70
type <- (type), 70
type <-, array-method (type), 70
type <-, vector-method (type), 70
union, 4, 58
union (sets), 58
union, GenomicRanges, GenomicRanges-method, 59
unique, 4, 72, 72
unique, Rle-method, 72
unlist, 4, 73, 73
unlist, List-method, 73
unsplit, 4, 74, 74
unsplit, List-method, 74
UnstitchedIPos, 61
unstrand (strand), 62
unz-class (S3-classes-as-S4-classes), 56
updateObject, 5, 75
updateObject, ANY-method (updateObject), 75
updateObject, environment-method (updateObject), 75
updateObject, envRefClass-method (updateObject), 75
updateObject, formula-method (updateObject), 75
updateObject, list-method (updateObject), 75
updateObjectFromSlots (updateObject), 75
updateObjectTo, 76
url-class (S3-classes-as-S4-classes), 56
validObject, 75, 76
var, 77, 77
Vector, 8, 23, 51, 60, 67
weights, 5, 78, 78
weights, PLMset-method, 78
which, 4, 79, 79
which, DelayedArray-method, 79
which.max, 4
which.max (which.min), 80
which.max, NumericList-method, 80
which.min, 4, 80, 80
width, 4
width (start), 60
width <- (start), 60
xtabs, 5, 81, 81, 82
xtabs, DataFrame-method, 82