

Biobase

November 11, 2009

R topics documented:

abstract ........................................ 2
addVigs2WinMenu ................................ 3
Aggregate .......................................... 4
annotatedDataFrameFrom-methods ............... 5
annotation ....................................... 5
anyMissing ..................................... 6
assayData ....................................... 7
Biobase-package .................................. 7
bioReposList .................................... 8
cache ........................................... 9
channelNames ................................... 10
channel ........................................... 11
aggregator ....................................... 11
AnnotatedDataFrame ......................... 12
AssayData-class ............................... 14
class:characterORMIAME ..................... 16
container ....................................... 16
eSet ............................................. 17
ExpressionSet ................................... 20
MIAME ........................................... 23
MultiSet ......................................... 25
NChannelSet-class .............................. 27
SnpSet ........................................... 29
VersionedBiobase .............................. 31
Versioned ....................................... 32
classVersion ................................... 33
VersionsNull ................................... 34
Versions ......................................... 35
combine ......................................... 36
contents ......................................... 38
copyEnv .......................................... 39
copySubstitute .................................. 40
createPackage ................................... 42
data:aaMap ..................................... 43
data:geneData .................................. 44
reporter ......................................... 44
data:sample.ExpressionSet .................... 45
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>data:sample.MultiSet</td>
</tr>
<tr>
<td>Deprecated</td>
</tr>
<tr>
<td>description</td>
</tr>
<tr>
<td>dims</td>
</tr>
<tr>
<td>dumpPackTxt</td>
</tr>
<tr>
<td>esApply</td>
</tr>
<tr>
<td>exprs</td>
</tr>
<tr>
<td>featureData</td>
</tr>
<tr>
<td>featureNames</td>
</tr>
<tr>
<td>getPkgVigs</td>
</tr>
<tr>
<td>abstract</td>
</tr>
<tr>
<td>isCurrent</td>
</tr>
<tr>
<td>isUnique</td>
</tr>
<tr>
<td>isVersioned</td>
</tr>
<tr>
<td>lcSuffix</td>
</tr>
<tr>
<td>listLen</td>
</tr>
<tr>
<td>makeDataPackage</td>
</tr>
<tr>
<td>matchpt</td>
</tr>
<tr>
<td>multiassign</td>
</tr>
<tr>
<td>note</td>
</tr>
<tr>
<td>notes</td>
</tr>
<tr>
<td>openPDF</td>
</tr>
<tr>
<td>openVignette</td>
</tr>
<tr>
<td>package.version</td>
</tr>
<tr>
<td>phenoData</td>
</tr>
<tr>
<td>read.AnnotatedDataFrame</td>
</tr>
<tr>
<td>readExpressionSet</td>
</tr>
<tr>
<td>read.MIAME</td>
</tr>
<tr>
<td>reverseSplit</td>
</tr>
<tr>
<td>rowMedians</td>
</tr>
<tr>
<td>rowQ</td>
</tr>
<tr>
<td>ScalarObject-class</td>
</tr>
<tr>
<td>selectChannels</td>
</tr>
<tr>
<td>selectSome</td>
</tr>
<tr>
<td>storageMode</td>
</tr>
<tr>
<td>strbreak</td>
</tr>
<tr>
<td>subListExtract</td>
</tr>
<tr>
<td>testBioCConnection</td>
</tr>
<tr>
<td>updateObject</td>
</tr>
<tr>
<td>updateOldESet</td>
</tr>
<tr>
<td>userQuery</td>
</tr>
<tr>
<td>validMsg</td>
</tr>
</tbody>
</table>

Index 82
abstract

Retrieve Meta-data from eSets and ExpressionSets.

Description

These generic functions access generic data, abstracts, PubMed IDs and experiment data from instances of the eSet-class or ExpressionSet-class.

Usage

abstract(object)
pubMedIds(object)
pubMedIds(object) <- value
experimentData(object)
experimentData(object) <- value

Arguments

object Object, possibly derived from eSet-class or MIAME-class
value Value to be assigned; see class of object (e.g., eSet-class) for specifics.

Value

abstract returns a character vector containing the abstract (as in a published paper) associated with object.
pubMedIds returns a character vector of PUBMED IDs associated with the experiment.
experimentData returns an object representing the description of an experiment, e.g., an object of MIAME-class

Author(s)

Biocore

See Also

ExpressionSet-class, eSet-class, MIAME-class

addVigs2WinMenu

Add Menu Items to an Existing/New Menu of Window

Description

This function adds a menu item for a package’s vignettes.

Usage

addVigs2WinMenu(pkgName)
Aggregate

Arguments

pkgName

pkgName - a character string for the name of an R package

Details

The original functions addVig2Menu, addVig4Win, addVig4Unix, addNonExisting, addPDF2Vig have been replaced by addVigs2WinMenu, please use those instead.

Value

The functions do not return any value.

Author(s)

Jianhua Zhang and Jeff Gentry

Examples

# Only works for windows now
if(interactive() && .Platform$OS.type == "windows" &&
    .Platform$GUI == "Rgui"){
    addVigs2WinMenu("Biobase")
}

Aggregate

A Simple Aggregation Mechanism.

Description

Given an environment and an aggregator (an object of class aggregate simple aggregations are made.

Usage

Aggregate(x, agg)

Arguments

x

The data to be aggregated.

agg

The aggregator to be used.

Details

Given some data, x the user can accumulate (or aggregate) information in env using the two supplied functions. See the accompanying documentation for a more complete example of this function and its use.

Value

No value is returned. This function is evaluated purely for side effects. The symbols and values in env are altered.
annotatedDataFrameFrom-methods

Author(s)
R. Gentleman

See Also
new.env, class: aggregator

Examples

agg1 <- new("aggregator")
Aggregate(letters[1:10], agg1)
# the first 10 letters should be symbols in env1 with values of 1
Aggregate(letters[5:11], agg1)
# now letters[5:10] should have value 2
bb <- mget(letters[1:11], env=aggenv(agg1), ifnotfound=NA)
t1 <- as.numeric(bb); names(t1) <- names(bb)
t1
# a b c d e f g h i j k
# 1 1 1 1 2 2 2 2 2 2 1

annotatedDataFrameFrom-methods

Methods for Function annotatedDataFrameFrom in Package
'Biobase'

Description

annotatedDataFrameFrom is a convenience for creating AnnotatedDataFrame objects.

Methods

Use the method with annotatedDataFrameFrom(object, byrow=FALSE, ...); the argument byrow must be specified.

signature(object="assayData") This method creates an AnnotatedDataFrame using sample (when byrow=FALSE) or feature (byrow=TRUE) names and dimensions of an AssayData object as a template.

signature(object="matrix") This method creates an AnnotatedDataFrame using column (when byrow=FALSE) or row (byrow=TRUE) names and dimensions of a matrix object as a template.

signature(object="NULL") This method (called with ‘NULL’ as the object) creates an empty AnnotatedDataFrame; provides dimLabels based on value of byrow.

Author(s)
Biocore team
### annotation

**Annotate eSet data.**

#### Description
This generic function handles methods for adding and retrieving ‘annotation’ and ‘description’ information for eSets. An annotation is the name of the file describing the chip used for the experiment.

#### Usage

```r
annotation(object)
annotation(object) <- "hgu95av2"
```

#### Arguments

- `object`: Object derived from class eSet

#### Value
`annotation(object)` returns a character vector indicating the annotation package.

#### Author(s)
Biocore

#### See Also
- `eSet-class`, `ExpressionSet-class`, `SnpSet-class`

### anyMissing

**Checks if there are any missing values in an object or not**

#### Description
Checks if there are any missing values in an object or not.

#### Usage

```r
anyMissing(x=NULL)
```

#### Arguments

- `x`: A vector.

#### Details
The implementation of this method is optimized for both speed and memory.
assayData

Value
Returns TRUE if a missing value was detected, otherwise FALSE.

Author(s)
Henrik Bengtsson (http://www.braju.com/R/)

Examples
x <- rnorm(n=1000)
x[seq(300,length(x),by=100)] <- NA
stopifnot(anyMissing(x) == any(is.na(x)))

assayData Retrieve assay data from eSets and ExpressionSets.

Description
This generic function accesses assay data stored in an object derived from the eSet or ExpressionSet class.

Usage
assayData(object)
assayData(object) <- value

Arguments
object Object derived from class eSet
value Named list or environment containing one or more matrices with identical dimensions

Value
assayData applied to eSet-derived classes returns a list or environment; applied to ExpressionSet, the method returns an environment. See the class documentation for specific details.

Author(s)
Biocore

See Also
eSet-class, ExpressionSet-class, SnpSet-class
Biobase-package  Biobase Package Overview

Description

Biobase Package Overview

Details

Important data classes: ExpressionSet, AnnotatedDataFrame MIAME. Full help on methods and associated functions is available from within class help pages.

Additional data classes: eSet, MultiSet. Additional manipulation and data structuring classes: Versioned, VersionedBiobase, aggregator, container.

Vignette routines: openVignette, getPkgVigs, openPDF.

Package manipulation functions: createPackage and package.version

Data sets: aaMap, sample.ExpressionSet, geneData.

Introductory information is available from vignettes, type openVignette().

Full listing of documented articles is available in HTML view by typing help.start() and selecting Biobase package from the Packages menu or via library(help="Biobase").

Author(s)

O. Sklyar

biocReposList  Return a list of Bioconductor package repositories

Description

This function returns a named character vector of Bioconductor package repositories. The vector can be used as the repos argument to install.packages and friends.

Usage

biocReposList()

Details

The repository URLs are hardcoded for each release.
Value

- bioc: URL of main Bioc package repository
- aData: URL for Bioc annotation data package repository
- eData: URL for Bioc experiment data package repository
- oh: URL for Bioc Omegahat package repository. This repository contains the versions of Omegahat packages that were tested with the current Bioc release.
- li: URL for Bioc Lindsey package repository.
- cran: URL for Bioc CRAN package repository. This is just a normal CRAN repository.

Author(s)

S. Falcon

Examples

brl <- biocReposList()

---

cache

Evaluate an expression if its value is not already cached.

Description

Cache the evaluation of an expression in the file system.

Usage

cache(expr, dir=".", prefix="tmp_R_cache_", name)

Arguments

- expr: An expression of the form LHS <- RHS. Where LHS is a variable name, RHS is any valid expression, and <- must be used (= will not work).
- dir: A string specifying the directory into which cache files should be written (also where to go searching for an appropriate cache file).
- prefix: A string giving the prefix to use when naming and searching for cache files. The default is "tmp_R_cache_"
- name: Unused. This argument is present as a compatibility layer for the deprecated calling convention.

Details

This function can be useful during the development of computationally intensive workflows, for example in vignettes or scripts. The function uses a cache file in dir which defaults to the current working directory whose name is obtained by paste(prefix, name, ".RData", sep=" ").

When cache is called and the cache file exists, it is loaded and the object whose name is given on the left of <- in expr is returned. In this case, expr is not evaluated.
When `cache` is called and the cache file does not exist, `expr` is evaluated, its value is saved into a cache file, and then its value is returned. The `expr` argument must be of the form of `someVar <- {expressions}`. That is, the left hand side must be a single symbol name and the next syntactic token must be `<-`. To flush the cache and force recomputation, simply remove the cache files. You can use `file.remove` to do this.

**Value**

The (cached) value of `expr`.

**Note**

The first version of this function had a slightly different interface which is now deprecated (but still functional). The old version has arguments `name` and `expr` and the intended usage is: `foo <- cache("foo", expr)`.

**Author(s)**

Wolfgang Huber, ⟨huber@ebi.ac.uk⟩ Seth Falcon, ⟨sfalcon@fhcrc.org⟩

**Examples**

```r
bigCalc <- function() runif(10)
cache(myComplicatedObject <- bigCalc())
aCopy <- myComplicatedObject
remove(myComplicatedObject)
cache(myComplicatedObject <- bigCalc())
stopifnot(all.equal(myComplicatedObject, aCopy))
allCacheFiles <- list.files(
    .,
    pattern="^tmp_R_cache_.*\.RData$",
    full.name=TRUE)
file.remove(allCacheFiles)
```

**channelNames**

Retrieve channel names from object

**Description**

This generic function reports the channels present in an object.

**Usage**

```r
channelNames(object, ...)```

**Arguments**

- `object` An S4 object, typically derived from class `eSet`
- `...` Additional argument, not currently used.

**Value**

`character`.
Author(s)

Biocore

Examples

```r
obj <- new("NChannelSet",
    R=matrix(runif(100), 20, 5),
    G=matrix(runif(100), 20, 5))
channelNames(obj)
```

---

**channel**

*Create a new ExpressionSet instance by selecting a specific channel*

Description

This generic function extracts a specific element from an object, returning an instance of the ExpressionSet class.

Usage

```r
channel(object, name, ...)
```

Arguments

- `object`: An S4 object, typically derived from class `eSet`
- `name`: The name of the channel, a (length one) character vector.
- `...`: Additional arguments.

Value

An instance of class `ExpressionSet`.

Author(s)

Biocore

Examples

```r
obj <- new("NChannelSet",
    R=matrix(runif(100), 20, 5),
    G=matrix(runif(100), 20, 5))
## G channel as ExpressionSet
channel(obj, "G")
```
**Description**

A class of objects designed to help aggregate calculations over an iterative computation. The aggregator consists of three objects. An environment to hold the values. A function that sets up an initial value the first time an object is seen. An aggregate function that increments the value of an object seen previously.

**Details**

This class is used to help aggregate different values over function calls. A very simple example is to use leave one out cross-validation for prediction. At each stage we first perform feature selection and then cross-validate. To keep track of how often each feature is selected we can use an aggregator. At the end of the cross-validation we can extract the names of the features chosen from `aggenv`.

**Creating Objects**

```r
new('aggregator', aggenv = [environment], initfun = [function], aggfun = [function])
```

**Slots**

- `aggenv`: Object of class `environment`, holds the values between iterations
- `initfun`: Object of class `function` specifies how to initialize the value for a name the first time it is encountered
- `aggfun`: Object of class `function` used to increment (or perform any other function) on a name

**Methods**

- `aggenv(aggregator)`: Used to access the environment of the aggregator
- `aggfun(aggregator)`: Used to access the function that aggregates
- `initfun(aggregator)`: Used to access the initializer function

**See Also**

`Aggregate`
AnnotatedDataFrame Class Containing Measured Variables and Their Meta-Data Description.

Description

An AnnotatedDataFrame consists of two parts. There is a collection of samples and the values of variables measured on those samples. There is also a description of each variable measured. The components of an AnnotatedDataFrame can be accessed with pData and varMetadata.

Extends

Versioned

Creating Objects

```r
new("AnnotatedDataFrame")
new("AnnotatedDataFrame", data=data.frame(), varMetadata=data.frame(),
dimLabels=c("rowNames", "columnNames"))
```

AnnotatedDataFrame instances are created using new. The initialize method takes up to three arguments, data, varMetadata, and dimLabels. data is a data.frame of the samples (rows) and measured variables (columns). varMetadata is a data.frame with the number of rows equal to the number of columns of the data argument. varMetadata describes aspects of each measured variable. dimLabels provides aesthetic control for labeling rows and columns in the show method. varMetadata and dimLabels can be missing.

```r
as(data.frame, "AnnotatedDataFrame")
```

coerces a data.frame to an AnnotatedDataFrame.

```r
annotatedDataFrameFrom may be a convenient way to create an AnnotatedDataFrame from AssayData-class.
```

Slots

Class-specific slots:

- **data**: A data.frame containing samples (rows) and measured variables (columns).
- **varMetadata**: A data.frame with number of rows equal number of columns in data, and at least one column, named labelDescription, containing a textual description of each variable.

Methods

Class-specific methods.

```r
as(annotatedDataFrame, "data.frame")
```

Coerce objects of AnnotatedDataFrame to data.frame.

```r
combine(<AnnotatedDataFrame>, <AnnotatedDataFrame>)
```

Bind data from one AnnotatedDataFrame to a second AnnotatedDataFrame, returning the result as an AnnotatedDataFrame. Row (sample) names in each argument must be unique. Variable names present in both arguments occupy a single column in the resulting AnnotatedDataFrame. Variable names unique to either argument create columns with values assigned for those samples where the variable is present. varMetadata in the returned AnnotatedDataFrame is updated to reflect the combination.
AnnotatedDataFrame

pData(<AnnotatedDataFrame>, pData(<AnnotatedDataFrame>) <- <data.frame>:
Set and retrieve the data (samples and variables) in the AnnotatedDataFrame

varMetadata(<AnnotatedDataFrame>, varMetadata(<AnnotatedDataFrame>) <- <data.frame>:
Set and retrieve the meta-data (variables and their descriptions) in the AnnotatedDataFrame

featureNames(<AnnotatedDataFrame>, featureNames(<AnnotatedDataFrame>) <- <ANY>:
Set and retrieve the feature names in AnnotatedDataFrame; a synonym for sampleNames.

sampleNames(<AnnotatedDataFrame>, sampleNames(<AnnotatedDataFrame>) <- <ANY>:
Set and retrieve the sample names in AnnotatedDataFrame

varLabels(<AnnotatedDataFrame>, varLabels(<AnnotatedDataFrame>) <- <data.frame>:
Set and retrieve the variable labels in the AnnotatedDataFrame

dimLabels(<AnnotatedDataFrame>, dimLabels(<AnnotatedDataFrame>) <- <character>
Retrieve labels used for display of AnnotatedDataFrame, e.g., 'rowNames', 'column-Names'.

Standard generic methods:

initialize(<AnnotatedDataFrame>): Object instantiation, used by new; not to be called directly by the user.

as(<data.frame>, "AnnotatedDataFrame"):: Convert a data.frame to an AnnotatedDataFrame.

as(<phenoData>, <AnnotatedDataFrame>): Convert old-style phenoData-class objects to AnnotatedDataFrame, issuing warnings as appropriate.

validObject(<AnnotatedDataFrame>): Validity-checking method, ensuring coordination between data and varMetadata elements

updateObject(object, ..., verbose=FALSE) Update instance to current version, if necessary. See updateObject

isCurrent(object) Determine whether version of object is current. See isCurrent

isVersioned(object) Determine whether object contains a 'version' string describing its structure. See isVersioned

show(<AnnotatedDataFrame>) Abbreviated display of object

[[<variable>, <variable>]: Subset operation, taking two arguments and indexing the sample and variable. Returns an AnnotatedDataFrame, i.e., including relevant metadata. Unlike a data.frame, setting drop=TRUE generates an error.

[[<variable>, ...]] <- <new_value>, $<variable> <- <new_value>: Replace or add a variable to pData. ... can include named arguments (especially labelDescription) to be added to varMetadata.

dim(<AnnotatedDataFrame>, ncol(<AnnotatedDataFrame>): Number of samples and variables (dim) and variables (ncol) in the argument.

Author(s)

V.J. Carey, after initial design by R. Gentleman

See Also

eSet, ExpressionSet, read.AnnotatedDataFrame
Examples

```r
df <- data.frame(x=1:6,
                 y=rep(c("Low", "High"),3),
                 z=LETTERS[1:6],
                 row.names=paste("Sample", 1:6, sep="_"))
metaData <-
data.frame(labelDescription=c(
                "Numbers",
                "Factor levels",
                "Characters"))

new("AnnotatedDataFrame")
new("AnnotatedDataFrame", data=df)
new("AnnotatedDataFrame",
data=df, varMetadata=metaData)
as(df, "AnnotatedDataFrame")

obj <- new("AnnotatedDataFrame")
pData(obj) <- df
varMetadata(obj) <- metaData
validObject(obj)
```

---

**AssayData-class**  
*Class "AssayData"

**Description**

Container class defined as a class union of list and environment. Designed to contain one or more matrices of the same dimension.

**Methods**

- `combine` signature(x = "AssayData", y = "AssayData"): This method uses `cbind` to create new AssayData elements that contain the samples of both arguments x and y. Both AssayData arguments to combine must have the same collection of elements. The elements must have identical numbers of rows (features). The numerical contents of any columns (samples) present in the same element of different AssayData must be identical. The storage-Mode of the AssayData arguments must be identical, and the function returns an AssayData with storageMode matching the incoming mode. See also `combine,eSet,eSet-method`

- `featureNames` signature(object = "AssayData")
- `featureNames<-` signature(object = "AssayData", value = "ANY"): Return or set the feature names as a character vector. These are the row names of the AssayData elements. value can be a character or numeric vector; all entries must be unique.

- `sampleNames` signature(object = "AssayData")
- `sampleNames<-` signature(object = "AssayData", value="ANY"): Return or set the sample names. These are the column names of the the AssayData elements and the row names of phenoData. value can a character or numeric vector.

- `storageMode` signature(object = "AssayData")
storageMode<- signature(object = "AssayData", value="character"): Return or set the storage mode for the instance. value can be one of three choices: "lockedEnvironment", "environment", and "list". Environments offer a mechanism for storing data that avoids some of the copying that occurs when using lists. Locked environment help to ensure data integrity. Note that environments are one of the few R objects that are pass-by-reference. This means that if you modify a copy of an environment, you also modify the original. For this reason, we recommend using lockedEnvironment whenever possible.

Additional functions operating on AssayData include:

assayData[[name]] Select element name from assayData.

assayDataNew(storage.mode = c("lockedEnvironment", "environment", "list"), ...) Use storage.mode to create a new list or environment containing the named elements in . . .

assayDataValidMembers(assayData, required) Validate assayData, ensuring that the named elements required are present, matrices are of the same dimension, and featureNames (rownames) are consistent (identical or NULL) across entries.

assayDataElement(object, element) See eSet-class

assayDataElementReplace(object, element, value) See eSet-class

assayDataElementNames(object) See eSet-class

Author(s)

Biocore

See Also

eSet-class ExpressionSet-class
**container**

**A Lockable List Structure with Constraints on Content**

**Description**

Container class that specializes the list construct of R to provide content and access control

**Creating Objects**

```
new('container', x = [list], content = [character], locked = [logical])
```

**Slots**

- **x**  list of entities that are guaranteed to share a certain property
- **content**  tag describing container contents
- **locked**  boolean indicator of locked status. Value of TRUE implies assignments into the container are not permitted

**Methods**

Class-specific methods:

- `content(container)`  returns `content` slot of argument
- `locked(container)`  returns `locked` slot of argument

Standard methods defined for 'container':

- `show(container)`  prints container
- `length(container)`  returns number of elements in the container
- `[[index] and `[[index, value)`  access and replace elements in the container
- `[(index)`  make a subset of a container (which will itself be a container)

**Examples**

```
x1 <- new("container", x=vector("list", length=3), content="lm")
lm1 <- lm(rnorm(10)~runif(10))
x1[[1]] <- lm1
```
Class to Contain High-Throughput Assays and Experimental Metadata

Description

Container for high-throughput assays and experimental metadata. Classes derived from eSet contain one or more identical-sized matrices as assayData elements. Derived classes (e.g., ExpressionSet-class, SnpSet-class) specify which elements must be present in the assayData slot.

eSet object cannot be instantiated directly; see the examples for usage.

Creating Objects

eSet is a virtual class, so instances cannot be created.

Objects created under previous definitions of eSet-class can be coerced to classes derived based on eSet using updateOldESet.

Slots

Introduced in eSet:

- **assayData**: Contains matrices with equal dimensions, and with column number equal to nrow(phenoData).  
  Class: AssayData-class

- **phenoData**: Contains variables describing sample (i.e., columns in assayData) phenotypes.  
  Class: AnnotatedDataFrame-class

- **featureData**: Contains variables describing features (i.e., rows in assayData) unique to this experiment. Use the annotation slot to efficiently reference feature data common to the annotation package used in the experiment. Class: AnnotatedDataFrame-class

- **experimentData**: Contains details of experimental methods. Class: MIAME-class

- **annotation**: Label associated with the annotation package used in the experiment. Class: character

- **.classVersion**: Character vector describing the R and Biobase version number used to created the instance. Intended for developer use. Class: Versions-class

Methods

Methods defined in derived classes (e.g., ExpressionSet-class, SnpSet-class) may override the methods described here.

Class-specific methods:

- **sampleNames(object) and sampleNames(object) <- value**: Coordinate accessing and setting sample names in assayData and phenoData

- **featureNames(object), featureNames(object) <- value**: Coordinate accessing and setting of feature names (e.g. genes, probes) in assayData.

- **dims(object)**: Access the common dimensions (dim) or column numbers (ncol), or dimensions of all members (dims) of assayData.
phenoData(object), phenoData(object) <- value: Access and set phenoData. Adding new columns to phenoData is often more easily done with eSetObject["columnName"] <- value.

pData(object), pData(object) <- value: Access and set sample data information. Adding new columns to pData is often more easily done with eSetObject["columnName"] <- value.

varMetadata(object), varMetadata(eSet, value) Access and set metadata describing variables reported in phenoData.

varLabels(object), varLabels(eSet, value) <-: Access and set variable labels in phenoData.

featureData(object), featureData(object) <- value: Access and set featureData.

fData(object), fData(object) <- value: Access and set feature data information.

fvarMetadata(object), fvarMetadata(eSet, value) Access and set metadata describing features reported in fData.

fvarLabels(object), fvarLabels(eSet, value) <-: Access and set variable labels in featureData.

assayData(object), assayData(object) <- value: signature(object = "eSet", value = "AssayData"): Access and replace the AssayData slot of an eSet instance. assayData returns a list or environment: elements in assayData not accessible in other ways (e.g., via exprs applied directly to the eSet) can most reliably be accessed with, e.g., assayData(obj)["se.exprs"].

experimentData(object), experimentData(object) <- value: Access and set details of experimental methods.

description(object), description(object) <- value: Synonymous with experimentData.

notes(object), notes(object) <- value: signature(object = "eSet", value = "list") Retrieve and set unstructured notes associated with eSet. signature(object = "eSet", value = "character") As with value = "list", but append value to current list of notes.

pubMedIds(object), pubMedIds(eSet, value) Access and set PMIDs in experimentData.

abstract(object): Access abstract in experimentData.

annotation(object), annotation(object) <- value Access and set annotation label indicating package used in the experiment.

preproc(object), preproc(object) <- value: signature(object = "eSet", value = "list") Access and set preprocessing information in the MIAME-class object associated with this eSet.

combine(eSet, eSet): Combine two eSet objects. To be combined, eSets must have identical numbers of featureNames, distinct sampleNames, and identical annotation.

storageMode(object), storageMode(eSet, character) <-: Change storage mode of assayData. Can be used to 'unlock' environments, or to change between list and environment modes of storing assayData.

Standard generic methods:

initialize(object): Object instantiation, can be called by derived classes but not usually by the user.

validObject(object): Validity-checking method, ensuring (1) all assayData components have the same number of features and samples; (2) the number and names of phenoData rows match the number and names of assayData columns.
as(eSet, "ExpressionSet") Convert instance of class "eSet" to instance of ExpressionSet-class, if possible.

as(eSet, "MultiSet") Convert instance of class "eSet" to instance of MultiSet-class, if possible.

updateObject(object, ..., verbose=FALSE) Update instance to current version, if necessary. Usually called through class inheritance rather than directly by the user. See updateObject

updateObjectTo(object, template, ..., verbose=FALSE) Update instance to current version by updating slots in template, if necessary. Usually call by class inheritance, rather than directly by the user. See updateObjectTo

isCurrent(object) Determine whether version of object is current. See isCurrent

isVersioned(object) Determine whether object contains a 'version' string describing its structure. See isVersioned

show(object) Informatively display object contents.

dim(object), ncol Access the common dimensions (dim) or column numbers (ncol), of all members (dims) of assayData.

object[(index)]: Conducts subsetting of matrices and phenoData components

object$name, object$name<-value Access and set name column in phenoData

object[[i, ...]], object[[i, ...]]<-value Access and set column i (character or numeric index) in phenoData. The ... argument can include named variables (especially labelDescription) to be added to varMetadata.

Additional functions:

assayDataElement(object, element) Return matrix element from assayData slot of object.

assayDataElement(object, element) <- value) Set element element in assayData slot of object to matrix value

assayDataElementReplace(object, element, value) Set element element in assayData slot of object to matrix value

assayDataElementNames(object) Return element names in assayData slot of object

updateOldESet Update versions of eSet constructed using listOrEnv as assayData slot (before May, 2006).

Author(s)

Biocore team

See Also

Method use in ExpressionSet-class. Related classes AssayData-class, AnnotatedDataFrame-class, MIAME-class. Derived classes ExpressionSet-class, SnpSet-class. To update objects from previous class versions, see updateOldESet.

Examples

# update previous eSet-like class oldESet to existing derived class
## Not run: updateOldESet(oldESet, "ExpressionSet")

# create a new, ad hoc, class, for personal use
ExpressionSet

# all methods outlined above are available automatically
setClass("MySet", contains="eSet")
new("MySet")

# Create a more robust class, with initialization and validation methods
# to ensure assayData contains specific matrices
setClass("TwoColorSet", contains="eSet")

setMethod("initialize", "TwoColorSet",
  function(.Object,
    phenoData = new("AnnotatedDataFrame"),
    experimentData = new("MIAME"),
    annotation = character(),
    R = new("matrix"),
    G = new("matrix"),
    Rb = new("matrix"),
    Gb = new("matrix"),
    ...
  ) {
    callNextMethod(.Object,
      phenoData = phenoData,
      experimentData = experimentData,
      annotation = annotation,
      R=R, G=G, Rb=Rb, Gb=Gb,
      ...
    )
  })

setValidity("TwoColorSet", function(object) {
  assayDataValidMembers(assayData(object), c("R", "G", "Rb", "Gb"))
})

new("TwoColorSet")

# eSet objects cannot be instantiated directly, only derived objects
try(new("eSet"))

removeClass("MySet")
removeClass("TwoColorSet")
removeMethod("initialize", "TwoColorSet")
Creating Objects

new("ExpressionSet")
new("ExpressionSet", phenoData = new("AnnotatedDataFrame"), featureData = new("AnnotatedDataFrame"), experimentData = new("MIAME"), annotation = character(0), exprs = new("matrix"))

This creates an ExpressionSet with assayData implicitly created to contain exprs. Additional named matrix arguments with the same dimensions as exprs are added to assayData; the row and column names of these additional matrices should match those of exprs.

new("ExpressionSet", assayData = assayDataNew(exprs=new("matrix")), phenoData = new("AnnotatedDataFrame"), featureData = new("AnnotatedDataFrame"), experimentData = new("MIAME"), annotation = character(0))

This creates an ExpressionSet with assayData provided explicitly. In this form, the only required named argument is assayData.

as([exprSet],"ExpressionSet")

ExpressionSet instances are usually created through new("ExpressionSet"), ...). Usually the arguments to new include exprs (a matrix of expression data, with features corresponding to rows and samples to columns), phenoData, featureData, experimentData, and annotation. phenoData, featureData, experimentData, and annotation can be missing, in which case they are assigned default values.

Slots

Inherited from eSet:

assayData: Contains matrices with equal dimensions, and with column number equal to nrow(phenoData). assayData must contain a matrix exprs with rows representing features (e.g., reporters) and columns representing samples. Additional matrices of identical size (e.g., representing measurement errors) may also be included in assayData. Class: AssayData-class
phenoData: See eSet
featureData: See eSet
experimentData: See eSet
annotation: See eSet

Methods

Class-specific methods.

as(exprSet,"ExpressionSet") Coerce objects of exprSet-class to ExpressionSet
as(object,"data.frame") Coerce objects of ExpressionSet-class to data.frame by transposing the expression matrix and concatenating phenoData
exprs(ExpressionSet), exprs(ExpressionSet,matrix) <- Access and set elements named exprs in the AssayData-class slot.
esApply(ExpressionSet, MARGIN, FUN, ...) 'apply'-like function to conveniently operate on ExpressionSet objects. See esApply.
write.exprs(ExpressionSet) Write expression values to a text file. It takes the same arguments as write.table

Derived from eSet:
updateObject(object, ..., verbose=FALSE)  
  Update instance to current version, if necessary. See updateObject and eSet

isCurrent(object)  
  Determine whether version of object is current. See isCurrent

isVersioned(object)  
  Determine whether object contains a 'version' string describing its structure. See isVersioned

assayData(ExpressionSet):  
  See eSet

sampleNames(ExpressionSet) and sampleNames(ExpressionSet)<-:  
  See eSet

featureNames(ExpressionSet), featureNames(ExpressionSet, value)<-:  
  See eSet

dims(ExpressionSet):  
  See eSet

phenoData(ExpressionSet), phenoData(ExpressionSet, value)<-:  
  See eSet

varLabels(ExpressionSet), varLabels(ExpressionSet, value)<-:  
  See eSet

varMetadata(ExpressionSet), varMetadata(ExpressionSet, value)<-:  
  See eSet

pData(ExpressionSet), pData(ExpressionSet, value)<-:  
  See eSet

varMetadata(ExpressionSet), varMetadata(ExpressionSet, value)  
  See eSet

experimentData(ExpressionSet), experimentData(ExpressionSet, value)<-:  
  See eSet

pubMedIds(ExpressionSet), pubMedIds(ExpressionSet, value)  
  See eSet

abstract(ExpressionSet):  
  See eSet

annotation(ExpressionSet), annotation(ExpressionSet, value)<-:  
  See eSet

combine(ExpressionSet, ExpressionSet):  
  See eSet

storageMode(ExpressionSet), storageMode(ExpressionSet, character)<-:  
  See eSet

Standard generic methods:

initialize(ExpressionSet):  
  Object instantiation, used by new; not to be called directly by the user.

updateObject(ExpressionSet):  
  Update outdated versions of ExpressionSet to their current definition. See updateObject, Versions-class.

validObject(ExpressionSet):  
  Validity-checking method, ensuring that exprs is a member of assayData. checkValidity(ExpressionSet) imposes this validity check, and the validity checks of eSet.

makeDataPackage(object, author, email, packageName, packageVersion, license, biocViews, filePath, description=paste(abstract(object), collapse="\n\n"), ...)  
  Create a data package based on an ExpressionSet object. See makeDataPackage.

as(exprSet, ExpressionSet):  
  Coerce exprSet to ExpressionSet.

as(eSet, ExpressionSet):  
  Coerce the eSet portion of an object to ExpressionSet.

show(ExpressionSet)  
  See eSet

dim(ExpressionSet), ncol  
  See eSet

ExpressionSet[(index):  
  See eSet

ExpressionSet$, ExpressionSet$<->  
  See eSet

ExpressionSet[[i]], ExpressionSet[[i]]<->  
  See eSet

Author(s)

Biocore team
MIAME

Class for Storing Microarray Experiment Information

Description

Class MIAME covers MIAME entries that are not covered by other classes in Bioconductor. Namely, experimental design, samples, hybridizations, normalization controls, and pre-processing information.

Slots

name: Object of class character containing the experimenter name
lab: Object of class character containing the laboratory where the experiment was conducted
contact: Object of class character containing contact information for lab and/or experimenter
title: Object of class character containing a single-sentence experiment title
abstract: Object of class character containing an abstract describing the experiment
url: Object of class character containing a URL for the experiment
samples: Object of class list containing information about the samples
hybridizations: Object of class list containing information about the hybridizations
normControls: Object of class list containing information about the controls such as housekeeping genes
preprocessing: Object of class list containing information about the pre-processing steps used on the raw data from this experiment
pubMedIds: Object of class character listing strings of PubMed identifiers of papers relevant to the dataset
other: Object of class list containing other information for which none of the above slots does not applies

Methods
Class-specific methods:
abstract(MIAME): An accessor function for abstract.
combine(MIAME,MIAME): Combine two objects of MIAME-class, issuing warnings when ambiguities encountered.
expinfo(MIAME): An accessor function for name, lab, contact, title, and url.
hybridizations(MIAME): An accessor function for hybridizations.
normControls(MIAME): An accessor function for normControls.
notes(MIAME), notes(MIAME) <- value: Accessor functions for other. notes(MIAME) <- character appends character to notes; use notes(MIAME) <- list to replace the notes entirely.
otherInfo(MIAME): An accessor function for other.
preproc(MIAME): An accessor function for preprocessing.
pubMedIds(MIAME), pubMedIds(MIAME) <- value: Accessor function for pubMedIds.
samples(MIAME): An accessor function for samples.

Standard generic methods:
updateObject(object, ..., verbose=FALSE) Update instance to current version, if necessary. See updateObject
isCurrent(object) Determine whether version of object is current. See isCurrent
isVersioned(object) Determine whether object contains a 'version' string describing its structure. See isVersioned
show(MIAME): Renders information about the MIAME information

Author(s)
Rafael A. Irizarry

References
http://www.mged.org/Workgroups/MIAME/miame_1.1.html

See Also
class:characterORMIAME, read.MIAME
MultiSet

Class to Contain and Describe High-Throughput Expression Level Assays.

Description

Container for high-throughput assays and experimental metadata. MultiSet is derived from eSet-class. MultiSet differs from ExpressionSet-class because MultiSet can contain any element(s) in assayData (ExpressionSet must have an element named exprs).

Extends

Directly extends class eSet.

Creating Objects

new('MultiSet', phenoData = [AnnotatedDataFrame], experimentData = [MIAME], annotation = [character], ...)
updateOldESet(oldESet,"MultiSet")

MultiSet instances are usually created through new("MultiSet", ...). The ... arguments to new are matrices of expression data (with features corresponding to rows and samples to columns), phenoData, experimentData, and annotation. phenoData, experimentData, and annotation can be missing, in which case they are assigned default values.

updateOldESet will take a serialized instance (e.g., saved to a disk file with save object created with earlier definitions of the eSet-class, and update the object to MultiSet. Warnings are issued when direct translation is not possible; incorrectly created oldESet instances may not be updated.

Slots

Inherited from eSet:

assayData: Contains zero or more matrices with equal dimensions, and with column number equal to nrow(phenoData). Each matrix in assayData has rows representing features (e.g., reporters) and columns representing samples. Class: AssayData-class

phenoData: See eSet-class

experimentData: See eSet-class

annotation: See eSet-class

Methods

Class-specific methods: none

Derived from eSet-class:

updateObject (object, ..., verbose=FALSE) Update instance to current version, if necessary. See updateObject and eSet

isCurrent (object) Determine whether version of object is current. See isCurrent

isVersioned (object) Determine whether object contains a ‘version’ string describing its structure. See isVersioned
sampleNames(MultiSet) and sampleNames(MultiSet)<-: See eSet-class
featureNames(MultiSet), featureNames(MultiSet, value)<-: See eSet-class
dims(MultiSet): See eSet-class
phenoData(MultiSet), phenoData(MultiSet, value)<-: See eSet-class
varLabels(MultiSet), varLabels(MultiSet, value)<-: See eSet-class
varMetadata(MultiSet), varMetadata(MultiSet, value)<-: See eSet-class
pData(MultiSet), pData(MultiSet, value)<-: See eSet-class
varMetadata(MultiSet), varMetadata(MultiSet, value) See eSet-class
experimentData(MultiSet), experimentData(MultiSet, value)<-: See eSet-class
pubMedIds(MultiSet), pubMedIds(MultiSet, value) See eSet-class
abstract(MultiSet) See eSet-class
annotation(MultiSet), annotation(MultiSet, value)<- See eSet-class
combine(MultiSet, MultiSet): See eSet-class
storageMode(eSet), storageMode(eSet, character)<-: See eSet-class

Standard generic methods:

initialize(MultiSet): Object instantiation, used by new; not to be called directly by the user.
validObject(MultiSet): Validity-checking method, ensuring that all elements of assayData are matrices with equal dimensions.
as(eSet, MultiSet): Coerce the eSet portion of an object to MultiSet.
show(MultiSet) See eSet-class
dim(MultiSet), ncol See eSet-class
MultiSet[(index)]: See eSet-class
MultiSet$, MultiSet$<- See eSet-class

Author(s)

Biocore team

See Also

eSet-class, ExpressionSet-class

Examples

# create an instance of ExpressionSet
new("MultiSet")
NChannelSet-class

Class to contain data from multiple channel array technologies

Description

Container for high-throughput assays and experimental meta-data. Data are from experiments where a single ‘chip’ contains several (more than 1) different ‘channels’. All channels on a chip have the same set of ‘features’. An experiment consists of a collection of several N-channel chips; each chip is a ‘sample’.

An NChannelSet provides a way to coordinate assay data (expression values) with phenotype information and references to chip annotation data; it extends the eSet class.

An NChannelSet allows channels to be extracted (using the channels method, mentioned below), and subsets of features or samples to be selected (using [features], <samples>). Selection and subsetting occur so that relevant phenotypic data is maintained by the selection or subset.

Objects from the Class

Objects can be created by calls of the form new("NChannelSet", assayData, phenoData, ...). See the examples below.

Slots

**assayData**: Object of class AssayData, usually an environment containing matrices of identical size. Each matrix represents a single channel. Columns in each matrix correspond to samples, rows to features. Once created, NChannelSet manages coordination of samples and channels.

**phenoData**: Object of class AnnotatedDataFrame.

The data component of the AnnotatedDataFrame is data.frame with number of rows equal to the number of samples. Columns of the data component correspond to measured covariates.

The varMetadata component consists of mandatory columns labelDescription (providing a textual description of each column label in the data component) and channel. The channel of varMetadata is a factor, with levels equal to the names of the assayData channels, plus the special symbol _ALL_. The channel column is used to indicate which channel(s) the corresponding column in the data component of AnnotatedDataFrame correspond; the _ALL_ symbol indicates that the data column is applicable to all channels.

varMetadata may contain additional columns with arbitrary information.

Once created, NChannelSet coordinates selection and subsetting of channels in phenoData.

**featureData**: Object of class AnnotatedDataFrame, used to contain feature data that is unique to this experiment; feature-level descriptions common to a particular chip are usually referenced through the annotation slot.

**experimentData**: Object of class MIAME containing descriptions common to a particular chip are usually referenced through the annotation slot.

**annotation**: Object of class "character". Usually a length-1 character string identifying the chip technology used during the experiment. The annotation string is used to retrieve information about features, e.g., using the annotation package.

**__classVersion__**: Object of class Versions, containing automatically created information about the class definition Biobase package version, and other information about the user system at the time the instance was created. See classVersion and updateObject for examples of use.
NChannelSet-class

Extends

Class "eSet", directly. Class "VersionedBiobase", by class "eSet", distance 2. Class "Versioned", by class "eSet", distance 3.

Methods

Methods with class-specific functionality:

channel(object, name, ...) signature(object="NChannelSet", name="character"). Return an ExpressionSet created from the channel and corresponding phenotype of argument name. name must have length 1. Arguments ... are rarely used, but are passed to the ExpressionSet constructor, for instance to influence storage.mode.

channelNames(object) signature(object = "NChannelSet"). Obtain names of channels contained in object.

selectChannels(object, names, ...) signature(object = "NChannelSet", names = "character"). Create a new NChannelSet from object, containing only channels in names. The ... is not used by this method.

object[features, samples] signature(object = "NChannelSet", features = "ANY", samples = "ANY"). Create a new NChannelSet from object, containing only elements matching features and samples; either index may be missing, or a character, numeric, or logical vector.

sampleNames(object) <- value signature(object = "NChannelSet", value = "list") assign each (named) element of value to the sampleNames of the correspondingly named elements of assayData in object.

Methods with functionality derived from eSet: annotation, assayData, assayData<-, classVersion, classVersion<-, dim, dims, experimentData, featureData, phenoData, phenoData<-, pubMedIds, sampleNames, sampleNames<-, storageMode, varMetadata, isCurrent, isVersioned, updateObject.

Additional methods: coerce ('as', to convert between objects, if possible), initialize (used internally for creating objects), show (invoked automatically when the object is displayed to the screen)

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

See Also

eSet, ExpressionSet.

Examples

## An empty NChannelSet
objc <- new("NChannelSet")

## An NChannelSet with two channels (R, G) and no phenotypic data
objc <- new("NChannelSet",
          R=matrix(0,10,5), G=matrix(0,10,5))

## An NChannelSet with two channels and channel-specific phenoData
R <- matrix(0, 10, 3, dimnames=list(NULL, LETTERS[1:3]))
G <- matrix(1, 10, 3, dimnames=list(NULL, LETTERS[1:3]))
assayData <- assayDataNew(R=R, G=G)
```
data <- data.frame(ChannelRData=numeric(ncol(R)),
                   ChannelGData=numeric(ncol(R)),
                   ChannelRAndG=numeric(ncol(R)))
varMetadata <- data.frame(labelDescription=c(
  "R-specific phenoData",
  "G-specific phenoData",
  "Both channel phenoData"),
  channel=factor(c("R", "G", "_ALL_")))
phenoData <- new("AnnotatedDataFrame",
                 data=data, varMetadata=varMetadata)
obj <- new("NChannelSet",
           assayData=assayData, phenoData=phenoData)
obj
```

### SnpSet

**Class to Contain Objects Describing High-Throughput SNP Assays.**

**Description**

Container for high-throughput assays and experimental metadata. SnpSet class is derived from eSet, and requires matrices call, callProbability as assay data members.

**Extends**

Directly extends class eSet.

**Creating Objects**

```
new('SnpSet', phenoData = [AnnotatedDataFrame], experimentData = [MIAME],
    annotation = [character], call = [matrix], callProbability = [matrix],
    ...
)
```

SnpSet instances are usually created through `new("SnpSet", ...)`. Usually the arguments to `new` include call (a matrix of genotypic calls, with features (SNPs) corresponding to rows and samples to columns), phenoData, experimentData, and annotation. phenoData, experimentData, and annotation can be missing, in which case they are assigned default values.

**Slots**

Inherited from eSet:

**assayData**: Contains matrices with equal dimensions, and with column number equal to `nrow(phenoData)`. assayData must contain a matrix call with rows representing features (e.g., SNPs) and columns representing samples, and a matrix callProbability describing the certainty of
the call. The content of call and callProbability are not enforced by the class. Additional matrices of identical size may also be included in assayData. Class: AssayData-class

phenoData: See eSet
experimentData: See eSet
annotation: See eSet

Methods

Class-specific methods:

exprs(SnpSet), exprs(SnpSet, matrix) <- Access and set elements named call in the AssayData slot.

Derived from eSet:

updateObject(object, ... , verbose=FALSE) Update instance to current version, if necessary. See updateObject and eSet
isCurrent(object) Determine whether version of object is current. See isCurrent
isVersioned(object) Determine whether object contains a 'version' string describing its structure. See isVersioned

sampleNames(SnpSet) and sampleNames(SnpSet) <-: See eSet
featureNames(SnpSet), featureNames(SnpSet, value) <-: See eSet
dims(SnpSet): See eSet

phenoData(SnpSet), phenoData(SnpSet, value) <-: See eSet
varLabels(SnpSet), varLabels(SnpSet, value) <-: See eSet
varMetadata(SnpSet), varMetadata(SnpSet, value) <-: See eSet
pData(SnpSet), pData(SnpSet, value) <-: See eSet
varMetadata(SnpSet), varMetadata(SnpSet, value) See eSet
experimentData(SnpSet), experimentData(SnpSet, value) <-: See eSet
pubMedIds(SnpSet), pubMedIds(SnpSet, value) See eSet

abstract(SnpSet): See eSet

annotation(SnpSet), annotation(SnpSet, value) <- See eSet
combine(SnpSet, SnpSet): See eSet

storageMode(eSet), storageMode(eSet, character) <-: See eSet

Standard generic methods:

initialize(SnpSet): Object instantiation, used by new; not to be called directly by the user.
validObject(SnpSet): Validity-checking method, ensuring that call and callProbability is a member of assayData. checkValidity(SnpSet) imposes this validity check, and the validity checks of eSet.

show(SnpSet) See eSet
dim(SnpSet), ncol See eSet
SnpSet[(index): See eSet
SnpSet$, SnpSet$ <- See eSet
VersionedBiobase

Author(s)

V.J. Carey, after initial design by R. Gentleman

See Also

eSet, ExpressionSet

VersionedBiobase  Class "VersionedBiobase"

Description

Use this class as a ‘superclass’ for classes requiring information about versions. By default, the class contains versions for R and Biobase. See Versioned-class for additional details.

Methods

set Versioned-class for methods.

Author(s)

Biocore

See Also

Versioned-class

Examples

```r
obj <- new("VersionedBiobase")
classVersion(obj)

obj <- new("VersionedBiobase", versions=list(A="1.0.0"))
classVersion(obj)

setClass("A", contains="VersionedBiobase")
classVersion("A")
a <- new("A")
classVersion(a)

obj <- new("VersionedBiobase", versions=c(MyVersion="1.0.0"))
classVersion(obj)

setClass("B",
    contains="VersionedBiobase",
    prototype=prototype(new("VersionedBiobase", versions=list(B="1.0.0"))))
classVersion("B")
b <- new("B")
classVersion(b)
```
**Versioned**

removeClass("A")
removeClass("B")

---

**Versioned**

**Class "Versioned"**

**Description**

Use this class as a ‘superclass’ for classes requiring information about versions.

**Methods**

The following are defined; package developers may write additional methods.

- `new("Versioned", ..., versions=list())` Create a new Versioned-class instance, perhaps with additional named version elements (the contents of `versions`) added. Named elements of `versions` are character strings that can be coerced using `package_version`, or `package_version` instances.

- `classVersion(object)` Obtain version information about instance `object`. See `classVersion`.

- `classVersion(object) <- value` Set version information on instance `object` to `value`; useful when `object` is an instance of a class that contains `VersionClass`. See `classVersion`.

- `classVersion(object)["id"] <- value` Create or update version information "id" on instance `object` to `value`; useful when `object` is an instance of a class that contains `VersionClass`. See `classVersion`.

- `show(object)` Default method returns `invisible`, to avoid printing confusing information when your own class does not have a `show` method defined. Use `classVersion(object)` to get or set version information.

**Author(s)**

Biocore

**See Also**

`Versions-class`

**Examples**

```r
obj <- new("Versioned", versions=list(A="1.0.0"))
obj
classVersion(obj)

setClass("A", contains="Versioned")

classVersion("A")
a <- new("A")
a # 'show' nothing by default
classVersion(a)

setClass("B",
````
contains="Versioned",
prototype=prototype(new("Versioned",versions=list(B="1.0.0"))),

classVersion("B")
b <- new("B")
classVersion(b)

classVersion(b)["B"] <- "1.0.1"
classVersion(b)
classVersion("B")

classVersion("B") < classVersion(b)
classVersion(b) == "1.0.1"

setClass("C",
representation(x="numeric"),
contains=\("VersionedBiobase\),
prototype=prototype(new("VersionedBiobase", versions=c(C="1.0.1"))),

setMethod("show", signature(object="C"),
function(object) print(object@x))

c <- new("C",x=1:10)
classVersion(c)

classVersion Retrieve information about versioned classes

Description

These generic functions return version information for classes derived from Versioned-class,
or VersionsNull-class for unversioned objects. The version information is an object of Versions-class.

By default, classVersion has the following behaviors:

**classVersion(Versioned-instance)** Returns a Versions-class object obtained from the object.

**classVersion("class")** Consults the definition of class and return the current version information, if available.

**classVersion(ANY)** Return a VersionsNull-class object to indicate no version information available.

By default, the classVersion<- method has the following behavior:

**classVersion(Versioned-instance)["id"] <- value** Assign (update or add) value to Versions-instance. value is coerced to a valid version description. see Versions-class for additional access methods.
Usage
classVersion(object)
classVersion(object) <- value

Arguments

object Object whose version is to be determined, as described above.
value Version-class object to assign to object of Versioned-class object.

Value
classVersion returns an instance of Versions-class

Author(s)
Biocore team

See Also
Versions-class

Examples

obj <- new("VersionedBiobase")

classVersion(obj)
classVersion(obj)["Biobase"]
classVersion(1:10) # no version
classVersion("ExpressionSet") # consult ExpressionSet prototype

classVersion(obj)["MyVersion"] <- "1.0.0"
classVersion(obj)

---

VersionsNull  Class "VersionsNull"

Description
A class used to represent the ‘version’ of unversioned objects. Useful primarily for method dispatch.

Methods
The following are defined; package developers may write additional methods.

new("VersionsNull", ...) Create a new VersionsNull-class instance, ignoring any additional arguments.

show(object) Display “No version”.

Author(s)
Biocore
See Also

classVersion

Examples

```r
obj <- new("VersionsNull")
obj

obj <- new("VersionsNull", A="1.0.0")  # warning
obj
```

Versions

Class "Versions"

Description

A class to record version number information. This class is used to report versions; to add version informa-
tion to your own class, use Versioned-class.

Methods

The following are defined; package developers may write additional methods.

```r
new("Versions", ...) Create a new Versions-class instance, perhaps with named version elements (the contents of ...) added. Named elements of versions are character strings that can be coerced using package_version, or package_version instances, Versions-class objects.
object["id"] Obtain version information "id" from object.
object["id"] <- value Create or update version information "id" on instance object.
object[["id"]]) Obtain version information "id" from object. The result is a list of integers, corresponding to entries in the version string.
object[["id"]]) <- value Create or update version information "id" on instance object.
object$id Obtain version information "id" from object. The result is a list of integers, corresponding to entries in the version string.
object$id <- value Create or update version information "id" on instance object.
show(object) Display version information.
updateObject(object) Update object to the current Versions-class representation. Note that this does not update another class that uses Versions-class to track the class version.
as(object, "character") Convert object to character representation, e.g., 1.0.0
object1 < object2 Compare object1 and object2 using version class information. Symbols in addition to < are admissable; see ?Ops
```

Author(s)

Biocore
**combine**

*Methods for Function combine in Package ‘Biobase’*

**Description**

This 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.

**See Also**

`classVersion`, `isCurrent`, `isVersioned`
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

**`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 for AnnotatedDataFrame, AssayData, MIAME, and eSet classes and subclasses.

Author(s)

Biocore

See Also

`merge`

Examples

```r
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])
```

contents

Function to retrieve contents of environments

Description

The contents method is used to retrieve the values stored in an environment.

Usage

contents(object, all.names)

Arguments

object The environment (data table) that you want to get all contents from
all.names a logical indicating whether to copy all values in as.list.environment

Value

A named list is returned, where the elements are the objects stored in the environment. The names of the elements are the names of the objects.

The all.names argument is identical to the one used in as.list.environment.

Author(s)

R. Gentleman

See Also

as.list.environment

Examples

z <- new.env()
multiassign(letters, 1:26, envir=z)
contents(z)
copyEnv

List-Environment interactions

Description

These functions can be used to make copies of environments, or to get/assign all of the objects inside of an environment.

Usage

```r
copyEnv(oldEnv, newEnv, all.names=FALSE)
l2e(vals, envir)
```  

Arguments

- **oldEnv**: An environment to copy from  
- **newEnv**: An environment to copy to. If missing, a new environment with the same parent environment as oldEnv.  
- **envir**: An environment to get/set values to. For `l2e` this can be left missing and a new environment of an appropriate size will be returned.  
- **vals**: A named list of objects to assign into an environment. The names must not include `NA` or `""` and should be unique.  
- **all.names**: Whether to retrieve objects with names that start with a dot.

Details

`l2e`: This function takes a named list and assigns all of its elements into an environment (using the names to name the objects). Unless you have an existing environment which you want to reuse, it is best to omit the `envir` argument. This way, the function will create a new environment with an efficient initial size. If the names of `vals` are not unique, a warning will be raised. The returned environment will contain the value associated with the last occurrence of any given duplicated name.  
`copyEnv`: This function will make a copy of the contents from `oldEnv` and place them into `newEnv`.

Author(s)

Jeff Gentry and R. Gentleman

See Also

`environment`, `as.list`

Examples

```r
z <- new.env(hash=TRUE, parent=emptyenv(), size=29L)
multiassign(c("a","b","c"), c(1,2,3), z)

a <- copyEnv(z)
ls(a)

q <- as.list(z)
```
Copy files, directory trees or between connections and replace all occurrences of a symbol by the corresponding value.

Usage

```
copySubstitute(src, dest, symbolValues, symbolDelimiter="@", allowUnresolvedSymbols=FALSE, recursive = FALSE, removeExtension = "\.in")
```

Arguments

- `src` Source, either a character vector with filenames and/or directory names, or a connection object.
- `dest` Destination, either a character vector of length 1 with the name of an existing, writeable directory, or a connection object. The class of the `dest` argument must match that of the `src` argument.
- `symbolValues` A named list of character strings.
- `symbolDelimiter` A character string of length one with a single character in it.
- `allowUnresolvedSymbols` Logical. If `FALSE`, then the function will execute `stop` if it comes across symbols that are not defined in `symbolValues`.
- `recursive` Logical. If `TRUE`, the function works recursively down a directory tree (see details).
- `removeExtension` Character. Matches to this regular expression are removed from filenames and directory names.

Details

Symbol substitution: this is best explained with an example. If the list `symbolValues` contains an element with name `FOO` and value `bar`, and `symbolDelimiter` is `@`, then any occurrence of `@FOO@` is replaced by `bar`. This applies both the text contents of the files in `src` as well as to the filenames. See examples.

If `recursive` is `FALSE`, both `src` and `dest` must be connection or a filenames. The text in `src` is read through the function `readLines`, symbols are replaced by their values, and the result is written to `dest` through the function `writeLines`.

If `recursive` is `TRUE`, `copySubstitute` works recursively down a directory tree (see details and example). `src` must be a character vector with multiple filenames or directory names, `dest` a directory name.

One use of this function is in `createPackage` for the automatic generation of packages from a template package directory.
**copySubstitute**

**Value**

None. The function is called for its side effect.

**Author(s)**

Wolfgang Huber [http://www.dkfz.de/mga/whuber](http://www.dkfz.de/mga/whuber)

**Examples**

```r
## create an example file
infile = tempfile()
outfile = tempfile()

writeLines(text=c("We will perform in @WHAT@:",
    "So, thanks to @WHOM@ at once and to each one,
    "Whom we invite to see us crown'd at @WHERE@."),
    con = infile)

## create the symbol table
z = list(WHAT="measure, time and place", WHOM="all", WHERE="Scone")

## run copySubstitute
copySubstitute(infile, outfile, z)

## display the results
readLines(outfile)
```

```r
##--------------------------------------------------------------
## This is a slightly more complicated example that demonstrates
## how copySubstitute works on nested directories
##--------------------------------------------------------------
```

```r
d = tempdir()
my.dir.create = function(x) {dir.create(x); return(x)}

unlink(file.path(d, "src")), recursive=TRUE)
unlink(file.path(d, "dest")), recursive=TRUE)

## create some directories and files:
src = my.dir.create(file.path(d, "src"))
dest = file.path(d, "dest")
d1 = my.dir.create(file.path(src, "dir1.in"))
d2 = my.dir.create(file.path(src, "dir2@FOO@.in"))
d3 = my.dir.create(file.path(d2, "dir3"))
d4 = my.dir.create(file.path(d3, "dir4"))
d5 = my.dir.create(file.path(d4, "dir5@BAR@"))

writeLines(c("File1:", "FOO: @FOO@"), file.path(d1, "file1.txt.in"))
writeLines(c("File2:", "BAR: @BAR@"), file.path(d2, "file2.txt.in"))
writeLines(c("File3:", "SUN: @SUN@"), file.path(d3, "file3.txt.in"))
writeLines(c("File4:", "MOON: @MOON@"), file.path(d4, "@SUN@.txt"))

## call copySubstitute
copySubstitute(src, dest, recursive=TRUE,
    symbolValues = list(FOO="thefoo", BAR="thebar",
        SUN="thesun", MOON="themoon"))
```
# createPackage

Create a Package Directory from a Template

## Description

Create a package directory from a template, with symbol-value substitution

## Usage

```r
createPackage(pkgname, destinationDir, originDir, symbolValues, unlink=FALSE, quiet=FALSE)
```

## Arguments

- **pkgname**
  - Character. The name of the package to be written.
- **destinationDir**
  - Character. The path to a directory where the package is to be written.
- **originDir**
  - Character. The path to a directory that contains the template package. Usually, this will contain a file named `DESCRIPTION`, and subdirectories `R`, `man`, `data`. In all files and filenames, symbols will be replaced by their respective values, see the parameter `symbolValues`.
- **symbolValues**
  - Named list of character strings. The symbol-to-value mapping. See `copySubstitute` for details.
- **unlink**
  - Logical. If `TRUE`, and `destinationDir` already contains a file or directory with the name `pkgname`, try to unlink (remove) it.
- **quiet**
  - Logical. If `TRUE`, do not print information messages.

## Details

The intended use of this function is for the automated mass production of data packages, such as the microarray annotation, CDF, and probe sequence packages. An example is in the function `makeProbePackage`.

No syntactic or other checking of the package is performed. For this, use R CMD check.

The symbols `@PKGNAME@` and `@DATE@` are automatically defined with the values of `pkgname` and \`\date\()`, respectively.

## Value

The function returns a list with one element `pkgdir`: the path to the package.

## Author(s)

Wolfgang Huber [http://www.dkfz.de/mga/whuber](http://www.dkfz.de/mga/whuber)
Dataset: Names and Characteristics of Amino Acids

The aaMap data frame has 20 rows and 6 columns. Includes elementary information about amino acids.

Usage

data(aaMap)

Format

This data frame contains the following columns:

name    amino acid name
let.1    one-letter code
let.3    three-letter code
scProp   side chain property at pH 7 (polar/nonpolar)
hyPhilic logical: side chain is hydrophilic at pH 7
acidic   logical: side chain is acidic at pH 7

Source

Nei M and Kumar S: Molecular evolution and phylogendetic (Oxford 2000), Table 1.2

Examples

data(aaMap)
**Description**

The `geneData` data frame has 500 rows and 26 columns. It consists of a subset of real expression data from an Affymetrix U95v2 chip. The data are anonymous. The covariate data `geneCov` and `geneCovariate` are made up. The standard error data `seD` is also made up.

**Usage**

```r
data(geneData)
```

**Format**

A 500 by 26 data frame.

**Source**

The J. Ritz Laboratory (S. Chiaretti).

**Examples**

```r
data(geneData)
data(geneCovariate)
data(seD)
```

---

**Description**

The `reporter` object is a 500 by 1 data frame. The rows represent the 500 probe ids in the `geneData` data. The values in `reporter` are the predefined probe types for the probes. `reporter` is used in conjunction with the `geneData` object and its associates.

**Usage**

```r
data(reporter)
```

**Format**

A 500 by 1 data frame
Details

There are 10 predefined probe types:

- **AFFX-** Quality Control (QC)
- **_f_** SequenceFamily
- **_g_** CommonGroups
- **_s_** SimilarityConstraint
- **_r_** RulesDropped
- **_i_** Incomplete
- **_b_** AmbiguousProbeSet
- **_l_** LongProbeSet
- **_at** AntiSenseTarget
- **_st** SenseTarget

Source


Examples

```r
data(reporter)
## maybe str(reporter) ; plot(reporter) ...
```

---

**data:sample.ExpressionSet**

*Dataset of class 'ExpressionSet'*

Description

The expression data are real but anonymized. The data are from an experiment that used Affymetrix U95v2 chips. The data were processed by dChip and then exported to R for analysis.

The data illustrate `ExpressionSet-class`, with `assayData` containing the required matrix element `exprs` and an additional matrix `se.exprs`. `se.exprs` has the same dimensions as `exprs`.

The phenoData and standard error estimates (se.exprs) are made up. The information in the "description" slot is fake.

Usage

```r
data(sample.ExpressionSet)
```

Format

The data for 26 cases, labeled A to Z and 500 genes. Each case has three covariates: sex (male/female); type (case/control); and score (testing score).

Examples

```r
data(sample.ExpressionSet)
```
**data:sample.MultiSet**

*Data set of class 'MultiSet'*

**Description**

The expression data are real but anonymized. The data are from an experiment that used Affymetrix U95v2 chips. The data were processed by dChip and then exported to R for analysis.

The phenoData, standard error estimates, and description data are fake.

**Usage**

```r
data(sample.MultiSet)
```

**Format**

The data for 4 cases, labeled a to d and 500 genes. Each case has five covariates: SlideNumber: number; FileName: name; Cy3: genotype labelled Cy3; Cy5: genotype labelled Cy5; Date: date.

**Examples**

```r
data(sample.MultiSet)
```

---

**Deprecated**

*Biobase Deprecated and Defunct*

**Description**

The function, class, or data object you have asked for has been deprecated or made defunct.

**description**

*Retrieve and set overall experimental information eSet-like classes.*

**Description**

These generic functions access experimental information associated with `eSet-class`.

**Usage**

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

**Arguments**

- `object` Object, possibly derived from class `eSet-class`.
- `value` Structured information describing the experiment, e.g., of `MIAME-class`.
- `...` Further arguments to be used by other methods.
Value

description returns an object of MIAME-class.

Author(s)

Biocore

See Also

eSet-class, MIAME-class

---

dims Retrieve dimensions of all elements in a list or environment

Description

This function returns the dimensions of element members in lists or environments such as AssayData-class.

Usage

dims(object)

Arguments

object List or environment object containing one or several matrices

Value

matrix of row and column dimensions, (in rows) for each element in object (columns).

Author(s)

Biocore

See Also

eSet-class
dumpPackTxt

Description
Dump textual description of a package

Usage
dumpPackTxt(package)

Arguments
package Character string naming an R package

Details
dumps DESCRIPTION and INDEX files from package sources

Value
stdout output

Note
Other approaches using formatDL are feasible

Author(s)
<stvjc@channing.harvard.edu>

Examples
dumpPackTxt("stats")

esApply
An apply-like function for ExpressionSet and related structures.

Description
esApply is a wrapper to apply for use with ExpressionSets. The application of a function to rows of an expression array usually involves variables in pData. esApply uses a special evaluation paradigm to make this easy. The function FUN may reference any data in pData by name.

Usage
esApply(X, MARGIN, FUN, ...)

esApply
An apply-like function for ExpressionSet and related structures.

Description
esApply is a wrapper to apply for use with ExpressionSets. The application of a function to rows of an expression array usually involves variables in pData. esApply uses a special evaluation paradigm to make this easy. The function FUN may reference any data in pData by name.

Usage
esApply(X, MARGIN, FUN, ...)
Arguments

X An instance of class ExpressionSet.
MARGIN The margin to apply to, either 1 for rows (samples) or 2 for columns (features).
FUN Any function
... Additional parameters for FUN.

Details

The pData from X is installed in an environment. This environment is installed as the environment of FUN. This will then provide bindings for any symbols in FUN that are the same as the names of the pData of X. If FUN has an environment already it is retained but placed after the newly created environment. Some variable shadowing could occur under these circumstances.

Value

The result of \( \text{with(pData}(x), \text{apply}(\text{exprs}(X), \text{MARGIN}, \text{FUN}, \text{...})) \).

Author(s)

V.J. Carey <stvjc@channing.harvard.edu>, R. Gentleman

See Also

apply, ExpressionSet

Examples

data(sample.ExpressionSet)
### sum columns of exprs
res <- esApply(sample.ExpressionSet, 1, sum)

### t-test, splitting samples by 'sex'
f <- function(x) {
  xx <- split(x, sex)
  t.test(xx[[1]], xx[[2]])$p.value
}
res <- esApply(sample.ExpressionSet, 1, f)

### same, but using a variable passed in the function call
f <- function(x, s) {
  xx <- split(x, s)
  mean(xx[[1]]) - mean(xx[[2]])
}
sex <- sample.ExpressionSet[["sex"]]
res <- esApply(sample.ExpressionSet, 1, f, s = sex)

# obtain the p-value of the t-test for sex difference
mytt.demo <- function(y) {
  ys <- split(y, sex)
  t.test(ys[[1]], ys[[2]])$p.value
}
sexPValue <- esApply(sample.ExpressionSet, 1, mytt.demo)
# obtain the p-value of the slope associated with score, adjusting for sex
# (if we were concerned with sign we could save the z statistic instead at coef[3,3]
myreg.demo <- function(y) {
  summary(lm(y ~ sex + score))$coef[3,4]
}
scorePValue <- esApply(sample.ExpressionSet, 1, myreg.demo)

# a resampling method
resamp <- function(ESET) {
  ntiss <- ncol(exprs(ESET))
  newind <- sample(1:ntiss, size = ntiss, replace = TRUE)
  ESET[newind,]
}

# a filter
q3g100filt <- function(eset) {
  apply(exprs(eset), 1, function(x) quantile(x,.75) > 100)
}

# filter after resampling and then apply
set.seed(123)
rest <- esApply({bool <- q3g100filt(resamp(sample.ExpressionSet)); sample.ExpressionSet[bool,]
  1, mytt.demo})

---

**exprs**

Retrieve expression data from eSets.

**Description**

These generic functions access the expression and error measurements of assay data stored in an object derived from the **eSet-class**.

**Usage**

```r
exprs(object)
exprs(object) <- value
se.exprs(object)
se.exprs(object) <- value
```

**Arguments**

- **object**: Object derived from class eSet.
- **value**: Matrix with rows representing features and columns samples.

**Value**

- **exprs** returns a (usually large!) matrix of expression values; **se.exprs** returns the corresponding matrix of standard errors, when available.

**Author(s)**

Biocore
featureData

Retrieve information on features recorded in eSet-derived classes.

Description

These generic functions access feature data (experiment specific information about features) and feature meta-data (e.g., descriptions of feature covariates).

Usage

```r
featureData(object)
featureData(object) <- value
fData(object)
fData(object) <- value
fvarLabels(object)
fvarLabels(object) <- value
fvarMetadata(object)
fvarMetadata(object) <- value
```

Arguments

- `object` Object, possibly derived from `eSet-class` or `AnnotatedDataFrame-class`.
- `value` Value to be assigned to corresponding object.

Value

- `featureData` returns an object containing information on both variable values and variable meta-data. `fvarLabels` returns a character vector of measured variable names. `fData` returns a data frame with features as rows, variables as columns. `fvarMetadata` returns a data frame with variable names as rows, description tags (e.g., unit of measurement) as columns.

Author(s)

Biocore

See Also

- `eSet-class`,
featureNames

Retrieve feature and sample names from eSets.

Description

These generic functions access the feature names (typically, gene or SNP identifiers) and sample names stored in an object derived from the eSet-class.

Usage

featureNames(object)
featureNames(object) <- value
sampleNames(object)
sampleNames(object) <- value

Arguments

object Object, possibly derived from class eSet.
value Character vector containing feature or sample names.

Value

featureNames returns a (usually long!) character vector uniquely identifying each feature. sampleNames returns a (usually shorter) character vector identifying samples.

Author(s)

Biocore

See Also

ExpressionSet-class, SnpSet-class

getPkgVigs List Vignette Files for a Package

Description

This function will return a listing of all vignettes stored in a package’s doc directory.

Usage

getPkgVigs(package = NULL)

Arguments

package A character vector of packages to search or NULL. The latter is for all attached packages (in search()).
isCurrent

Value
A dataframe with columns `package`, `filename`, `title`.

Author(s)
Jeff Gentry, modifications by Wolfgang Huber.

See Also
`openVignette`

Examples
```r
z <- getPkgVigs()
z # and look at them
```

Description
Use `help.search("your keyword", package="Biobase")`.

isCurrent
Use version information to test whether class is current

Description
This generic function uses `Versioned-class` information to ask whether an instance of a class (e.g., read from disk) has current version information.

By default, `isCurrent` has the following behaviors:

- `isCurrent(Versioned-instance)` Returns a vector of logicals, indicating whether each version matches the current version from the class prototype.
- `isCurrent(ANY)` Returns `NA`, indicating that the version cannot be determined.
- `isCurrent(Versioned-instance, "class")` Returns a logical vector indicating whether version identifiers shared between `Versioned-instance` and "class" are current.

Starting with R-2.6 / Bioconductor 2.1 / Biobase 1.15.1, `isCurrent(Versioned-instance, ...)` returns an element `S4` indicating whether the class has the ‘S4’ bit set; a value of `FALSE` indicates that the object needs to be recreated.

Usage
`isCurrent(object, value)`

Arguments
- `object` Object whose version is to be determined, as described above.
- `value` (Optional) character string identifying a class with which to compare versions.
isUnique

Value

isCurrent returns a logical vector.

Author(s)

Biocore team

See Also

Versions-class

Examples

```
obj <- new("VersionedBiobase")
isCurrent(obj)

isCurrent(1:10) # NA

setClass("A", contains="VersionedBiobase",
    prototype=prototype(new("VersionedBiobase", versions=c(A="1.0.0"))))

a <- new("A")
classVersion(a)

isCurrent(a, "VersionedBiobase") # is the 'VersionedBiobase' portion current?
classVersion(a)["A"] <- "1.0.1"
classVersion(a)
isCurrent(a, "VersionedBiobase")
isCurrent(a) # more recent, so does not match 'current' defined by prototype
removeClass("A")
```

isUnique Determine Unique Elements

Description

Determines which elements of a vector occur exactly once.

Usage

isUnique(x)

Arguments

x a vector

Value

A logical vector of the same length as x, in which TRUE indicates uniqueness.
Author(s)

Wolfgang Huber

See Also

unique, duplicated.

Examples

```r
x <- c(9:20, 1:5, 3:7, 0:8)
isUnique(x)
```

---

**isVersioned**

*Determine whether object or class contains versioning information*

Description

This generic function checks to see whether *Versioned-class* information is present. When the argument to *isVersioned* is a character string, the prototype of the class corresponding to the string is consulted.

By default, *isVersioned* has the following behaviors:

- `isVersioned(Versioned-instance)` Returns `TRUE` when the instance have version information.
- `isCurrent("class-name")` Returns `TRUE` when the named class extends *Versioned-class*.
- `isVersioned(ANY)` Returns `FALSE`

Usage

```r
isVersioned(object)
```

Arguments

- `object` Object or class name to check for version information, as described above.

Value

`isVersioned` returns a logical indicating whether version information is present.

Author(s)

Biocore team

See Also

*Versions-class*
Examples

```r
obj <- new("VersionedBiobase")
isVersioned(obj)

isVersioned(1:10) # FALSE

setClass("A", contains="VersionedBiobase",
        prototype=prototype(new("VersionedBiobase", versions=c(A="1.0.0"))))
a <- new("A")
isVersioned(a)

removeClass("A")
```

---

**lcSuffix**

*Compute the longest common prefix or suffix of a string*

**Description**

These functions find the longest common prefix or suffix among the strings in a character vector.

**Usage**

```r
lcPrefix(x, ignore.case=FALSE)
lcPrefixC(x, ignore.case=FALSE)
lcSuffix(x, ignore.case=FALSE)
```

**Arguments**

- `x`: a character vector.
- `ignore.case`: A logical value indicating whether or not to ignore the case in making comparisons.

**Details**

Computing the longest common suffix is helpful for truncating names of objects, like microarrays, that often have a common suffix, such as .CEL.

There are some potential problems with the approach used if multibyte character encodings are being used.

`lcPrefixC` is a faster implementation in C. It only handles ascii characters.

**Value**

The common prefix or suffix.

**Author(s)**

R. Gentleman

**See Also**

`nchar`
**Examples**

```r
s1 <- c("ABC.CEL", "DEF.CEL")
lcSuffix(s1)

s2 <- c("ABC.123", "ABC.456")
lcPrefix(s2)

CHK <- stopifnot

CHK(".CEL" == lcSuffix(s1))
CHK("bc" == lcSuffix(c("abc", "333abc", "bc")))
CHK("c" == lcSuffix(c("c", "abc", "xxxc")))
CHK("" == lcSuffix(c("c", "abc", "xxx")))

CHK("ABC." == lcPrefix(s2))
CHK("ab" == lcPrefix(c("abcd", "abcd123", "ab", "abc", "abc333333")))
CHK("a" == lcPrefix(c("abcd", "abcd123", "ax")))
CHK("a" == lcPrefix(c("a", "abcd123", "ax")))
CHK("" == lcPrefix(c("a", "abc", "xxx")))

CHK("ab" == lcPrefixC(c("abcd", "abcd123", "ab", "abc", "abc333333")))
CHK("a" == lcPrefixC(c("abcd", "abcd123", "ax")))
CHK("a" == lcPrefixC(c("a", "abcd123", "ax")))
CHK("" == lcPrefixC(c("a", "abc", "xxx")))
```

---

**listLen**

**Lengths of list elements**

**Description**

This function returns an integer vector with the length of the elements of its argument, which is expected to be a list.

**Usage**

```r
listLen(x)
```

**Arguments**

- `x` A list

**Details**

This function returns a vector of the same length as the list `x` containing the lengths of each element. The current implementation is intended for lists containing vectors and the C-level length function is used to determine length. This means no dispatch is done for the elements of the list. If your list contains S4 objects, you should use `sapply(x, length)` instead.

**Author(s)**

Jeff Gentry and R. Gentleman
makeDataPackage

See Also

sapply

Examples

```r
foo = lapply(1:8, rnorm)
listLen(foo)
```

---

**makeDataPackage**  
*Make an R package from a data object*

**Description**

This generic creates a valid R package from an R data object.

**Usage**

```r
makeDataPackage(object, author, email, 
packageName=deparse(substitute(object)), 
packageVersion=package_version("1.0.0"), 
license="Artistic-2.0", 
biocViews="ExperimentData", 
filePath=tempdir(), 
...)
```

**Arguments**

- `object`  
  An instance of an R data object.
- `author`  
  The author, as a character string.
- `email`  
  A valid email address for the maintainer, as a character string.
- `packageName`  
  The name of the package, defaults to the name of the object instance.
- `packageVersion`  
  The version number, as a character string.
- `license`  
  The license, as a character string.
- `biocViews`  
  A character vector of valid biocViews views.
- `filePath`  
  The location to create the package.
- `...`  
  Additional arguments to specific methods.

**Details**

The function makes use of various tools in R and Bioconductor to automatically generate the source files for a valid R package.

**Value**

The return value is that from a call to `link{createPackage}` which is invoked once the default arguments are set up. The data instance is stored in the data directory with a name the same as that of the resulting package.
Note

Developers implementing derived methods might force correct package name evaluation by including 'packageName' in any callNextMethod().

Author(s)

R. Gentleman

See Also

createPackage

Examples

data(sample.ExpressionSet)
## package created in tempdir()
s1 <- makeDataPackage(sample.ExpressionSet,
  author = "Foo Author",
  email = "foo@bar",
  packageName = "FooBarPkg",
  packageVersion = "1.0.0")

matchpt

Description

Find the nearest neighbors of a set of query points in the same or another set of points in an n-dimensional real vector space, using the Euclidean distance.

Usage

matchpt(x, y)

Arguments

x
A matrix (or vector) of coordinates. Each row represents a point in an ncol(x)-dimensional real vector space.

y
Optional, matrix (or vector) with the same number of columns as x.

details

If y is provided, the function searches for each point in x its nearest neighbor in y. If y is missing, it searches for each point in x its nearest neighbor in x, excluding that point itself. In the case of ties, only the neighbor with the smaller index is given.

The implementation is simple and of complexity nrow(x) times nrow(y). For larger problems, please consider one of the many more efficient nearest neighbor search algorithms.

Value

A data.frame with two columns and nrow(x) rows. The first column is the index of the nearest neighbor, the second column the distance to the nearest neighbor. If y was given, the index is a row number in y, otherwise, in x. The row names of the result are those of x.
multiassign

Author(s)

Oleg Sklyar ⟨osklyar@ebi.ac.uk⟩

Examples

```r
a <- matrix(c(2,2,3,5,1,8,-1,4,5,6), ncol=2L, nrow=5L)
rownames(a) = LETTERS[seq_len(nrow(a))]
matchpt(a)
b <- c(1,2,4,5,6)
d <- c(5.3, 3.2, 8.9, 1.3, 5.6, -6, 4.45, 3.32)
matchpt(b, d)
matchpt(d, b)
```

multiassign Assign Values to a Names

Description

Assign values to names in an environment.

Usage

```r
multiassign(x, value, envir = parent.frame(), inherits=FALSE)
```

Arguments

- `x`: A vector or list of names, represented by strings.
- `value`: a vector or list of values to be assigned.
- `envir`: the environment to use. See the details section.
- `inherits`: should the enclosing frames of the environment be inspected?

Details

The `pos` argument can specify the environment in which to assign the object in any of several ways: as an integer (the position in the `search` list); as the character string name of an element in the search list; or as an `environment` (including using `sys.frame` to access the currently active function calls). The `envir` argument is an alternative way to specify an environment, but is primarily there for back compatibility.

If `value` is missing and `x` has names then the values in each element of `x` are assigned to the names of `x`.

Value

This function is invoked for its side effect, which is assigning the `values` to the variables in `x`. If no `envir` is specified, then the assignment takes place in the currently active environment.

If `inherits` is `TRUE`, enclosing environments of the supplied environment are searched until the variable `x` is encountered. The value is then assigned in the environment in which the variable is encountered. If the symbol is not encountered then assignment takes place in the user’s workspace (the global environment).

If `inherits` is `FALSE`, assignment takes place in the initial frame of `envir`. 

Examples

```r
#-- Create objects 'r1', 'r2', ... 'r6' --
nam <- paste("r",1:6, sep=".")
multiassign(nam, 11:16)
ls(pat="^r..$")

#assign the values in y to variables with the names from y

y<-list(a=4,d=mean,c="aaa")
multiassign(y)
```

Informational Messages

Description

Generates an informational message that corresponds to its argument(s). Similar to warning() except prefaced by "Note:" instead of "Warning message:"

Usage

```r
note(...)```

Arguments

```r
... character vectors (which are pasted together) or NULL
```

Details

This function essentially `cat()`'s the created string to the screen. It is intended for messages to the user that are deemed to be 'informational', as opposed to warnings, etc.

Author(s)

Jeff Gentry

See Also

`warning`, `stop`

Examples

```r
note("This is an example of a note")
```
notes

Retrieve and set eSet notes.

Description

These generic functions access notes (unstructured descriptive data) associated with eSet-class. `notes(<ExpressionSet>) <- <character>` is unusual, in that the character vector is appended to the list of notes; use `notes(<ExpressionSet>) <- <list>` to entirely replace the list.

Usage

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

Arguments

- `object`: Object, possibly derived from class eSet-class.
- `value`: Character vector containing unstructured information describing the experiment.

Value

`notes` returns a list.

Author(s)

Biocore

See Also

`ExpressionSet-class`, `SnpSet-class`

openPDF

Open PDF Files in a Standard Viewer

Description

Displays the specified PDF file.

Usage

```r
openPDF(file, bg=TRUE)
```

Arguments

- `file`: A character string, indicating the file to view.
- `bg`: Should the pdf viewer be opened in the background.
openVignette

**Details**
Currently this function works on Windows and Unix platforms. Under Windows, whatever program is associated with the file extension will be used. Under Unix, the function will use the program named in the option "pdfviewer" (see `help(options)` for information on how this is set.) The `bg` argument is only interpreted on Unix.

**Value**
This function is executed for its side effects. The specified PDF file is opened in the PDF viewer and `TRUE` is returned.

**Author(s)**
Jeff Gentry

**Examples**

```r
## Not run: openPDF("annotate.pdf")
```

---

openVignette **Open a Vignette or Show Vignette Selection Menu**

**Description**
Using the data returned by `vignette` this function provides a simple easy to use interface for opening vignettes.

**Usage**

```r
openVignette(package=NULL)
```

**Arguments**

- `package` character string indicating the package to be used.

**Details**
If `package` is NULL then all packages are scanned for vignettes. The list of vignettes is presented to the user via the `menu` command. The user may select one of the vignettes to be opened in a PDF viewer.

**Value**
No value is returned; this function is run entirely for the side effect of opening the pdf document in the PDF viewer.

**Author(s)**
R. Gentleman
package.version  

Report Version of a Package

Description

Will report the version number of a requested installed package

Usage

package.version(pkg, lib.loc = NULL)

Arguments

pkg  
The name of the package

lib.loc  
a character vector describing the location of R library trees to search through, or ‘NULL’. The default value of ‘NULL’ corresponds to all libraries currently known.

Details

This function is a convenience wrapper around package.description, and will report simply the version number of the requested package. If the package does not exist or if the DESCRIPTION file can not be read, then an error will be thrown.

Value

A character string reporting the version number.

Author(s)

Jeff Gentry

See Also

package.description

Examples

package.version("Biobase")
**phenoData**

*Retrieve information on experimental phenotypes recorded in eSet and ExpressionSet-derived classes.*

**Description**

These generic functions access the phenotypic data (e.g., covariates) and meta-data (e.g., descriptions of covariates) associated with an experiment.

**Usage**

```r
phenoData(object)
phenoData(object) <- value
varLabels(object)
varLabels(object) <- value
varMetadata(object)
varMetadata(object) <- value
pData(object)
pData(object) <- value
```

**Arguments**

- `object`  
  Object, possibly derived from `eSet-class` or `AnnotatedDataFrame`.
- `value`  
  Value to be assigned to corresponding object.

**Value**

- `phenoData` returns an object containing information on both variable values and variable meta-data.
- `varLabels` returns a character vector of measured variables.
- `pData` returns a data frame with samples as rows, variables as columns.
- `varMetadata` returns a data frame with variable names as rows, description tags (e.g., unit of measurement) as columns.

**Author(s)**

Biocore

**See Also**

- `eSet-class`, `ExpressionSet-class`, `SnpSet-class`

---

**read.AnnotatedDataFrame**

*Read 'AnnotatedDataFrame'*

**Description**

Create an instance of class `AnnotatedDataFrame` by reading a file.
Usage

read.AnnotatedDataFrame(filename, path,
    sep = "\t", header = TRUE, quote = "", stringsAsFactors = FALSE,
    row.names = 1L,
    varMetadata.char="#",
    widget =getOption("BioC")$Base$use.widgets,
    sampleNames = character(0), ...)

Arguments

filename  file or connection from which to read.
path      (optional) directory in which to find filename.
row.names this argument gets passed on to read.table and will be used for the row
          names of the phenoData slot.
varMetadata.char
          lines beginning with this character are used for the varMetadata slot. See
          examples.
sep, header, quote, stringsAsFactors, ...
          further arguments that get passed on to read.table.
widget    logical. Currently this is not implemented, and setting this option to TRUE
          will result in an error. In a precursor of this function, read.phenoData, this
          option could be used to open an interactive GUI widget for entering the data.
sampleNames optional argument that could be used in conjunction with widget; do not use.

Details

The function read.table is used to read pData. The argument varMetadata.char is
passed on to that function as its argument comment.char. Lines beginning with varMetadata.char
are expected to contain further information on the column headers of pData. The format is of the
form: # variable: textual explanation of the variable, units, measurement
method, etc. (assuming that # is the value of varMetadata.char). See also examples.

Value

An instance of class AnnotatedDataFrame

Author(s)

Martin Morgan <mtmorgan@fhcrc.org> and Wolfgang Huber, based on read.phenoData by
Rafael A. Irizarry.

See Also

AnnotatedDataFrame for additional methods, read.table for details of reading in pheno-
typic data

Examples

exampleFile = system.file("extdata", "pData.txt", package="Biobase")

adf <- read.AnnotatedDataFrame(exampleFile)
readExpressionSet  

**Description**

Create an instance of class ExpressionSet by reading data from files. ‘widget’ functionality is not implemented for readExpressionSet.

**Usage**

```r
readExpressionSet(exprsFile, 
   phenoDataFile, 
   experimentDataFile, 
   notesFile, 
   path, 
   annotation, 
   ## arguments to read.* methods 
   exprsArgs=list(sep=sep, header=header, row.names=row.names, quote=quote, ...), 
   phenoDataArgs=list(sep=sep, header=header, row.names=row.names, quote=quote, stringsAsFactors=stringsAsFactors, ...), 
   experimentDataArgs=list(sep=sep, header=header, row.names=row.names, quote=quote, stringsAsFactors=stringsAsFactors, ...), 
   sep = "\t", header = TRUE, quote = "", stringsAsFactors = FALSE, row.names = ..., 
   ## widget 
   widget = getOption("BioC")$Base$use.widgets, 
   ...) 
```

**Arguments**

- `exprsFile` (character) File or connection from which to read expression values. The file should contain a matrix with rows as features and columns as samples. `read.table` is called with this as its `file` argument and further arguments given by `exprsArgs`.
- `phenoDataFile` (character) File or connection from which to read phenotypic data. `read.AnnotatedDataFrame` is called with this as its `file` argument and further arguments given by `phenoDataArgs`.
- `experimentDataFile` (character) File or connection from which to read experiment data. `read.MIAME` is called with this as its `file` argument and further arguments given by `experimentDataArgs`.
- `notesFile` (character) File or connection from which to read notes; `readLines` is used to input the file.
- `path` (optional) directory in which to find all the above files.
- `annotation` (character) A single character string indicating the annotation associated with this ExpressionSet.
- `exprsArgs` A list of arguments to be used with `read.table` when reading in the expression matrix.
read.MIAME

read.MIAME

Read MIAME Information into an Instance of Class 'MIAME'

Description

Reads MIAME information from a file or using a widget.

Usage

read.MIAME(filename = NULL, widget = getOption("BioC")$Base$use.widgets, ...)
reverseSplit

Arguments

filename
Filename from which to read MIAME information.

widget
Logical. If TRUE and a filename is not given, a widget is used to enter information.

... Further arguments to scan.

Details

Notice that the MIAME class tries to cover the MIAME entries that are not covered by other classes in Bioconductor. Namely, experimental design, samples, hybridizations, normalization controls, and pre-processing information.

The function scan is used to read. The file must be a flat file with the different entries for the instance of MIAME class separated by carriage returns. The order should be: name, lab, contact, title, abstract, and url.

Alternatively a widget can be used.

Value

An object of class MIAME.

Author(s)

Rafael Irizarry <rafa@jhu.edu>

See Also

MIAME, tkMIAME

Examples

miame <- read.MIAME(widget=FALSE) ##creates an empty instance
show(miame)

reverseSplit A function to reverse the role of names and values in a list.

Description

Given a list with names \( x \) and values in a set \( y \) this function returns a list with names in \( y \) and values in \( x \).

Usage

reverseSplit(inList)

Arguments

inList A named list with values that are vectors.
Details

First the list is unrolled to provide a two long vectors, names are repeated, once for each of their values. Then the names are split by the values.

This turns out to be useful for inverting mappings between one set of identifiers and an other.

Value

A list with length equal to the number of distinct values in the input list and values from the names of the input list.

Author(s)

R. Gentleman

See Also

split

Examples

l1 = list(a=1:4, b=c(2,3), d=c(4,5))
reverseSplit(l1)

rowMedians

Calculates the median for each row in a matrix

Description

Calculates the median for each row in a matrix.

Usage

rowMedians(imat, na.rm=FALSE)

Arguments

imat          A numeric matrix.
na.rm         If TRUE, NAs are excluded before calculating the medians, otherwise not.
...          Not use.

Value

Returns a double vector of length equal to number of rows in x.

Missing values

Missing values are excluded before calculating the medians.
Benchmarking

This implementation is optimized for speed and memory to calculate. As the example shows, this implementation is roughly 3-10 times faster than using `apply(x, MARGIN=1, FUN=medians)`. As the example might show, the `rowQ()` does not (have to) handle missing values, and is therefore in some cases faster.

Author(s)

Henrik Bengtsson

See Also

See `rowMeans()` in `colSums()`.

Examples

```r
set.seed(1)
x <- rnorm(n=234*543)
x[sample(1:length(x), size=0.1*length(x))] <- NA
dim(x) <- c(234,543)
y1 <- rowMedians(x, na.rm=TRUE)
y2 <- apply(x, MARGIN=1, FUN=median, na.rm=TRUE)
stopifnot(all.equal(y1, y2))
x <- cbind(x1=3, x2=c(4:1, 2:5))
stopifnot(all.equal(rowMeans(x), rowMedians(x)))
```

A function to compute empirical row quantiles.

Description

This function computes the requested quantile for each row of a matrix, or of an `ExpressionSet`.

Usage

```r
rowQ(imat, which)
rowMax(imat)
rowMin(imat)
```

Arguments

- `imat` Either a matrix or an `ExpressionSet`.
- `which` An integer indicating which order statistic should be returned.

Details

`rowMax`, `rowMin` and `rowMedians` simply call `rowQ` with the appropriate argument set.

The argument `which` takes values between 1, for the minimum per row, and `ncol(imat)`, for the maximum per row.
**ScalarObject-class**

**Value**

A vector of length equal to the number of rows of the input matrix containing the requested quantiles.

**Author(s)**

R. Gentleman

**See Also**

`rowMedians`, `rowMeans()` in `colSums()`.

**Examples**

data(sample.ExpressionSet)
rowMin(sample.ExpressionSet)
rowQ(sample.ExpressionSet, 4)

---

**ScalarObject-class**  Utility classes for length one (scalar) objects

**Description**

These classes represent scalar quantities, such as a string or a number and are useful because they provide their own validity checking. The classes `ScalarCharacter`, `ScalarLogical`, `ScalarInteger`, and `ScalarNumeric` all extend their respective base vector types and can be used interchangeably (except they should always have length one).

The `mkScalar` factory function provides a convenient way of creating `Scalar<type>` objects (see the examples section below).

**Usage**

`mkScalar(obj)`

**Arguments**

- `obj`  
  An object of type `character`, `logical`, `integer`, or `double`

**Author(s)**

Seth Falcon

**Examples**

```r
v <- list(mkScalar("a single string"),
          mkScalar(1),
          mkScalar(1L),
          mkScalar(TRUE))
sapply(v, class)
sapply(v, length)
```
selectChannels  

Create a new NChannelSet instance by selecting specific channels

Description
This generic function extracts specific elements from an object, returning a instance of that object.

Usage
selectChannels(object, names, ...)

Arguments
object  
An S4 object, typically derived from class eSet

names  
Character vector of named channels.

...  
Additional arguments.

Value
Instance of class object.

Author(s)
Biocore

Examples
obj <- new("NChannelSet",
    R=matrix(runif(100), 20, 5),
    G=matrix(runif(100), 20, 5))

## G channel as NChannelSet
selectChannels(obj, "G")

selectSome  

Extract elements of a vector for concise rendering

Description
Extract the first and last several elements of a vector for concise rendering; insert elipses to indiciated elided elements. This function is primarily meant for developer rather than end-user use.

Usage
selectSome(obj, maxToShow=5)

Arguments
obj  
A vector.

maxToShow  
The number of elements (including "...") to render.
storageMode

Details

This function can be used in ’show’ methods to give users exemplars of the tokens used in a vector. For example, an ExpressionSet built from a yeast experiment might have features enumerated using systematic gene names (e.g., YPR181C) or standard gene names (e.g., SEC23). The show method for ExpressionSet uses selectSome to alert the user to the tokens used, and thereby to indicate what vocabulary must be understood to work with the feature names.

Value

A string vector with at most maxToShow plus 1 elements, where an ellipsis ("...") is included to indicate incompleteness of the excerpt.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

Examples

selectSome(1:20)

storageMode

Retrieve or set storage mode for eSets.

Description

These generic functions report or change the storage mode used for assayData.

Usage

storageMode(object)
storageMode(object) <- value

Arguments

object Object, derived from class eSet
value Character vector containing "lockedEnvironment","environment", or "list". See AssayData-class for details.

Value

storageMode returns a length-1 character vector

Author(s)

Biocore

See Also

AssayData-class, eSet-class ExpressionSet-class, SnpSet-class
**strbreak**  
*Break Character Strings to Fit Width*

**Description**
Inserts line breaks (collapse) into input character strings. The main intention of this function is to prepare long strings for printing, so the output is not wider than width.

**Usage**

```
strbreak(x, width=getOption("width"), exdent=2, collapse="\n")
```

**Arguments**
- `x`: a character vector
- `width`: a positive integer giving the width of the output.
- `exdent`: a positive integer specifying the indentation of subsequent lines after the first line.
- `collapse`: a character. This is inserted to break lines.

**Author(s)**
Wolfgang Huber [http://www.ebi.ac.uk/huber](http://www.ebi.ac.uk/huber)

**See Also**
`strwrap, substring`

**Examples**

```r
longString = paste(rep(LETTERS, 10), collapse="", sep="")
cat(strbreak(longString))
```

---

**subListExtract**  
*Extract the same element from the sublists of a list*

**Description**
Given a list of lists, this function can be used to extract a named element from each sublist.

**Usage**

```
subListExtract(L, name, simplify = FALSE, keep.names = TRUE)
```
Arguments

L           A list of named lists
name        The name of the element in the sublists that should be extracted. This should be
            a length one character vector.
simplify    When TRUE, the return value will be an atomic vector. If any extracted sublist
            value has length not equal to one and simplify=TRUE, an error will be raised.
            When FALSE, a list is returned containing the extracted elements.
keep.names  If TRUE (default), the names of L will be attached to the returned vector.

Details

This function is implemented in C and is intended to be faster than calling lapply or sapply.

Value

If simplify=FALSE, a list will be returned having the same length as L, but with each element
containing the element named name from the corresponding inner list of L.

When simplify=TRUE, an atomic vector will be returned containing the extracted elements. If
any of the inner list elements do not have length one or cannot be put inside an atomic vector, an
error will be raised.

Author(s)

Seth Falcon

Examples

list_size = 500000
innerL = list(foo="foo", bar="bar")
L = rep(list(innerL), list_size)

system.time({j0 = sapply(L, function(x) x$foo)})
system.time({j1 = subListExtract(L, "foo", simplify=TRUE)})
stopifnot(all.equal(j0, j1))

LS = L[1:3]
names(LS) = LETTERS[1:3]
subListExtract(LS, "bar", simplify=TRUE)
subListExtract(LS, "bar", simplify=FALSE)
subListExtract(LS, "bar", simplify=TRUE, keep.names=FALSE)

Description

A function to check internet connectivity to Bioconductor

This function will attempt to determine if the user has internet connectivity to the Bioconductor
website. This is useful in many situations dealing with code that uses automated downloads and
other such things.
Usage

testBioCConnection()

Value

TRUE if a connection is possible, FALSE if not.

Author(s)

Jeff Gentry

Examples

z <- testBioCConnection()

updateObject

Update an object to its current class definition

Description

These generic functions return an instance of object updated to its current class definition (or to
the class definition of template, in the case of updateObjectTo).

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

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

updateObject (ANY, ..., verbose=FALSE) By default, updateObject uses heuris-
tic 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 link{updateObjectFromSlots}.

updateObject (list, ..., verbose=FALSE) Visit each element in list, applying updateObject (list
..., verbose=verbose).

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

updateObjectTo requires that the class of the returned object be the same as the class of the
template argument, and that the object is valid. Usually, updating proceeds by modifying slots
in template with information from object, and returning template. Use as to coerce an
object from one type to another; updateObjectTo might be useful to update a virtual superclass.
By default, updateObjectTo has the following behavior:

updateObjectTo (ANY-object, ANY-template) Attempt as (ANY-object, class (ANY-template)).
Sample methods are illustrated below.

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

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

**Usage**

```
updateObject (object, ..., verbose=FALSE)
updateObjectTo (object, template, ..., verbose=FALSE)
updateObjectFromSlots (object, objclass=class(object), ..., verbose=FALSE)
getObjectSlots (object)
```

**Arguments**

- `object` Object to be updated, or for slot information to be extracted from.
- `template` Instance representing a template for updating `object`.
- `objclass` Optional character string naming the class of the object to be created.
- `verbose` A logical, indicating whether information about the update should be reported. Use `message` to report this.
- `...` Additional arguments, for use in specific update methods.

**Value**

- `updateObject` returns a valid instance of `object`.
- `updateObjectTo` returns a valid instance of `template`.
- `updateObjectFromSlots` returns an instance of class `objclass`.
- `getObjectSlots` returns a list of named elements, with each element corresponding to a slot in `object`.

**Author(s)**

Biocore team

**See Also**

`Versions-class`

**Examples**

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

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

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

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

updateOldESet

Update previously created eSet object to current eSet structure

Description
This function updates eSet objects created in previous versions of Biobase to the current class structure. Warnings indicate when coercions change how data in the from object are altered. If the from object was not a valid object of the original eSet class, then updateOldESet may fail.

Usage
updateOldESet(from, toClass, ...)

Arguments
from Object created using a previous version of the eSet class.
toClass Character string identifying new class, e.g., "ExpressionSet"
... Additional arguments passed to the initialization method for class toClass

Value
Valid object of class toClass.

Author(s)
Biocore

See Also
eSet-class, ExpressionSet-class, SnpSet-class

Examples
## Not run:
updateOldESet(oldESet, "ExpressionSet")
## End(Not run)
userQuery

A function to query the user for input

Description

This function will output a given message and seek a response from the user, repeating the message until the input is from a valid set provided by the code.

Usage

userQuery(msg, allowed = c("y", "n"), default = "n", case.sensitive = FALSE)

Arguments

msg The output message
allowed Allowed input from the user
default Default response if called in batch mode
case.sensitive Is the response case sensitive? Defaults to FALSE

Value

The input from the user

Author(s)

Jeff Gentry

validMsg

Conditionally append result to validity message

Description

This function facilitates constructing messages during S4 class validation, and is meant for developer rather than end-user use.

Usage

validMsg(msg, result)

Arguments

msg A character vector or NULL.
result Any vector.

Details

This function appends result to msg, but only if result is a character vector.
Author(s)

Martin Morgan <mtmorgan@fherc.org>

Examples

```r
msg <- NULL
validMsg(msg, FALSE) # still NULL
msg <- validMsg(msg, "one")
validMsg(msg, "two")
```
Index

*Topic array
  cache, 7
  matchpt, 58

*Topic character
  strbreak, 74

*Topic classes
  aggregator, 10
  AnnotatedDataFrame, 11
  AssayData-class, 13
  class:characterORMIAME, 14
  container, 15
  eSet, 16
  ExpressionSet, 19
  MIAME, 22
  MultiSet, 24
  NChannelSet-class, 26
  ScalarObject-class, 71
  SnpSet, 28
  Versioned, 31
  VersionedBiobase, 30
  Versions, 34
  VersionsNull, 33

*Topic connection
  copySubstitute, 39

*Topic datasets
  data:aaMap, 42
  data:geneData, 43
  data:sample.ExpressionSet, 44
  data:sample.MultiSet, 45
  reporter, 43

*Topic data
  multiassign, 59

*Topic file
  read.AnnotatedDataFrame, 64
  read.MIAME, 67
  readExpressionSet, 66

*Topic interface
  addVigs2WinMenu, 2

*Topic internal
  abstract, 52
  Deprecated, 45

*Topic iteration
  anyMissing, 5

*Topic logic
  anyMissing, 5
  isUnique, 53

*Topic manip
  abstract, 1
  annotation, 4
  assayData, 5
  cache, 7
  channel, 9
  channelNames, 9
  classVersion, 32
  combine, 35
  contents, 37
  description, 45
  dims, 46
  exprs, 49
  featureData, 50
  featureNames, 51
  isCurrent, 52
  isUnique, 53
  isVersioned, 54
  lcSuffix, 55
  makeDataPackage, 57
  matchpt, 58
  notes, 61
  phenoData, 64
  read.AnnotatedDataFrame, 64
  readExpressionSet, 66
  reverseSplit, 68
  rowMedians, 69
  selectChannels, 72
  storageMode, 73
  subListExtract, 74
  updateObject, 76
  updateOldESet, 78

*Topic methods
  Aggregate, 2
  aggregator, 10
  annotatedDataFrameFrom-methods, 3
  container, 15
  esApply, 47

*Topic models
dumpPackTxt, 47
esApply, 47
* Topic package
  Biobase-package, 6
* Topic programming
  Aggregate, 2
copySubstitute, 39
createPackage, 41
* Topic utilities
  biocReposList, 7
copyEnv, 38
getPkgVigs, 51
listLen, 56
note, 60
openPDF, 61
openVignette, 62
package.version, 63
selectSome, 72
testBioCConnection, 75
userQuery, 79
validErrMsg, 79
[,AnnotatedDataFrame-method
  (AnnotatedDataFrame), 11
[,Versions-method(Versions), 34
[,container-method(container), 15
[,eSet-method(eSet), 16
[<-,Versions-method(Versions), 34
[[,AnnotatedDataFrame-method
  (AnnotatedDataFrame), 11
[[,container-method(container), 15
[<-,AnnotatedDataFrame-method
  (AnnotatedDataFrame), 11
[<-,eSet-method(eSet), 16
[<-,Versions-method(Versions), 34
[<-,container-method
  (container), 15
[<-,eSet-method(eSet), 16
$,AnnotatedDataFrame-method
  (AnnotatedDataFrame), 11
$,eSet-method(eSet), 16
$,Versions-method(Versions), 34
$,eSet-method(eSet), 16
anyMissing, 5
apply, 48
as, 76
as.data.frame.ExpressionSet
  (ExpressionSet), 19
as.list, 38
as.list.environment, 37
AssayData, 3, 26, 36
AssayData(AssayData-class), 13
assayData, 5, 17
assayData, AssayData-method
  (AssayData-class), 13
assayData, eSet-method(eSet), 16
AssayData-class, 11, 18, 73
agenv (abstract), 52
agenv, aggregator-method
  (aggregator), 10
aggfun (abstract), 52
aggfun, aggregator-method
  (aggregator), 10
Aggregate, 2, 11
aggregator, 6, 10
aggregator-class (aggregator), 10
AnnotatedDataFrame, 3, 6, 11, 26, 36,
  64, 65
AnnotatedDataFrame-class, 18, 50
AnnotatedDataFrame-class, 16
AnnotatedDataFrame-class
  (AnnotatedDataFrame), 11
annotatedDataFrameFrom, 11
annotatedDataFrameFrom
  (annotatedDataFrameFrom-methods), 3
annotatedDataFrameFrom,AssayData-method
  (annotatedDataFrameFrom-methods), 3
annotatedDataFrameFrom,matrix-method
  (annotatedDataFrameFrom-methods), 3
annotatedDataFrameFrom,NULL-method
  (annotatedDataFrameFrom-methods), 3
annotatedDataFrameFrom-methods, 3
annotatedDataset (Deprecated), 45
annotatedDataset-class
  (Deprecated), 45
annotation, 4
annotation,eSet-method(eSet), 16
annotation<-,annotation), 4
annotation<-,eSet,character-method
  (eSet), 16
anyMissing, 5
apply, 48
as, 76
as.data.frame.ExpressionSet
  (ExpressionSet), 19
as.list, 38
as.list.environment, 37
AssayData, 3, 26, 36
AssayData(AssayData-class), 13
assayData, 5, 17
assayData, AssayData-method
  (AssayData-class), 13
assayData, eSet-method(eSet), 16
AssayData-class, 11, 18, 73
aaMap, 6
aaMap (data:aaMap), 42
abstract, 1, 52
abstract,eSet-method(eSet), 16
abstract,MIAME-method(MIAME), 22
addVigs2WinMenu, 2
INDEX

ASSAYDATA-class, 13, 16, 20, 24, 29
assayData<- (assayData), 5
assayData<- eSet, ASSAYDATA-method (eSet), 16
assayDataElement (eSet), 16
assayDataElement<- (eSet), 16
assayDataElementNames (eSet), 16
assayDataNew (ASSAYDATA-class), 13
assayDataValidMembers (ASSAYDATA-class), 13

AssayData класс, 13, 16, 20, 24, 29
assayData <- assayData, 5
assayData <- eSet, AssayData-method (eSet), 16
assayDataElement (eSet), 16
assayDataElement<- (eSet), 16
assayDataElementNames (eSet), 16
assayDataNew (AssayData-class), 13
assayDataValidMembers (AssayData-class), 13

Biobase (Biobase-package), 6
Biobase-package, 6
biocReposList, 7

Biobase (Biobase-package), 6
Biobase-package, 6

cache, 7
channel, 9
channel, NChannelSet, character-method (NChannelSet-class), 26
channelNames, 9
channelNames, NChannelSet-method (NChannelSet-class), 26
characterORMIAME-class
class: characterORMIAME, 14
class.NChannelSet (NChannelSet-class), 26
class: aggregator, 3
class: aggregator (aggregator), 10
class: AnnotatedDataFrame (AnnotatedDataFrame), 11
class: annotatedDataset (Deprecated), 45
class: characterORMIAME, 14, 23
class: container (container), 15
class: eSet (eSet), 16
class: ExpressionSet (ExpressionSet), 19
class: exprList (Deprecated), 45
class: exprMatrix (Deprecated), 45
class: exprSet (Deprecated), 45
class: MIAME (MIAME), 22
class: MultiSet (MultiSet), 24
class: phenodata (Deprecated), 45
class: SnpSet (SnpSet), 28
classVersion, 26, 31, 32, 34, 35
classVersion, ANY-method (classVersion), 32
classVersion, character-method (classVersion), 32
classVersion, Versioned-method (Versioned), 31
classVersion<-(classVersion), 32
classVersion<-, Versioned, Versions-method (Versioned), 31
coerce, AnnotatedDataFrame, data.frame-method (AnnotatedDataFrame), 11
coerce, data.frame, AnnotatedDataFrame-method (AnnotatedDataFrame), 11
coerce, eSet, ExpressionSet-method (ExpressionSet), 19
coerce, eSet, MultiSet-method (MultiSet), 24
coerce, ExpressionSet, data.frame-method (ExpressionSet), 19
coerce, exprSet, ExpressionSet-method (ExpressionSet), 19
coerce, phenodata, AnnotatedDataFrame-method (AnnotatedDataFrame), 11
coerce, Versions, character-method (Versions), 34
colSums, 70, 71
combine, 35
combine, AnnotatedDataFrame, AnnotatedDataFrame-method (AnnotatedDataFrame), 11
combine, ANY, missing-method (combine), 35
combine, AssayData, AssayData-method (AssayData-class), 13
combine, data.frame, data.frame-method (combine), 35
combine, eSet, ANY-method (eSet), 16
combine, eSet, eSet-method (eSet), 16
combine, matrix, matrix-method (combine), 35
combine, MIAME, MIAME-method (MIAME), 22
Compare, character, Versions-method (Versions), 34
Compare, Versions, character-method (Versions), 34
Compare, Versions, Versions-method (Versions), 34
container, 6, 15
container-class (container), 15
content (abstract), 52
content, container-method (container), 15
contents, 37
contents, environment-method (abstract), 52
copyEnv, 38
copySubstitute, 39, 39, 41, 42
createPackage, 6, 39, 41, 58
data.frameOrNULL-class
data:geneCovariate
data:geneData
data:geneCovariate
data:geneCovariate
data:geneData
data:reporter
data:sample.eSet (Deprecated), 45
data:sample.expSet (Deprecated), 45
data:sample.MultiSet, 45
data:seD (data:geneData), 43
Deprecated, 45
description, 45
description, eSet-method (eSet), 16
description<-, (description), 45
description<-, eSet, MIAME-method (eSet), 16
df2pD (Deprecated), 45
dim, AnnotatedDataFrame-method
  (AnnotatedDataFrame), 11
dim, eSet-method (eSet), 16
dimLabels (AnnotatedDataFrame), 11
dimLabels, AnnotatedDataFrame-method
  (AnnotatedDataFrame), 11
dimLabels<-, (AnnotatedDataFrame), 11
dimLabels<-, AnnotatedDataFrame, character-method
  (AnnotatedDataFrame), 11
dims, 46
dims, eSet-method (eSet), 16
double, 69
dumpPackTxt, 47
duplicated, 54
eList (Deprecated), 45
eList, eSet-method (Deprecated), 45
eList<-, (Deprecated), 45
eList<-, eSet, AssayData-method
  (Deprecated), 45
environment, 38, 59
esApply, 20, 47
esApply, ExpressionSet-method
  (ExpressionSet), 19
eSet, 5, 6, 9, 13, 16, 19–21, 24, 26–30, 36, 50, 72
eSet-class, 1, 4, 6, 14, 22, 24, 25, 45, 46, 49–51, 61, 64, 73, 78
eSet-class (eSet), 16
experimentData (abstract), 1
experimentData, eSet-method (eSet), 16
experimentData<-(abstract), 1
experimentData<-, eSet, MIAME-method (eSet), 16
expinfo (abstract), 52
expinfo, MIAME-method (MIAME), 22
ExpressionSet, 5, 6, 10, 13, 19, 27, 30, 48, 50, 67, 73
ExpressionSet-class, 1, 4, 6, 14, 16, 18, 22, 24, 25, 44, 50, 51, 61, 64, 73, 78
ExpressionSet-class, 18, 20
ExpressionSet-class
  (ExpressionSet), 19
exprList (Deprecated), 45
exprList-class (Deprecated), 45
exprMatrix (Deprecated), 45
exprs, 49
exprs, eSet-method (eSet), 16
exprs, ExpressionSet-method
  (ExpressionSet), 19
exprs, SnpSet-method (SnpSet), 28
exprs<-(exprs), 49
exprs<-, eSet, AssayData-method
  (eSet), 16
exprs<-, ExpressionSet, matrix-method
  (ExpressionSet), 19
exprs<-, SnpSet, matrix-method
  (SnpSet), 28
exprSet (Deprecated), 45
exprSet-class, 20
exprSet-class (Deprecated), 45
factor, 36
FALSE, 5
fData (featureData), 50
fData, eSet-method (eSet), 16
fData<-(featureData), 50
fData<-, eSet, data.frame-method
  (eSet), 16
featureData, 50
featureData, eSet-method (eSet), 16
featureData<-(featureData), 50
featureData<-, eSet, AnnotatedDataFrame-method
  (eSet), 16
featureNames, 51
featureNames, AnnotatedDataFrame-method
  (AnnotatedDataFrame), 11
INDEX

featureNames, AssayData-method (AssayData-class), 13
featureNames, eSet-method (eSet), 16
featureNames <- featureNames, 51
featureNames <-, AnnotatedDataFrame-method (AnnotatedDataFrame), 11
featureNames <-, AssayData-method (AssayData-class), 13
featureNames <-, eSet-method (eSet), 16
file.remove, 8
geneCov (data:geneData), 43
geneCovariate (data:geneData), 43
geneData (data:geneData), 6
geneNames (data:geneData), 43
geneNames, ExpressionSet-method (Depreciated), 45
geneNames <- (Depreciated), 45
geneNames <- ExpressionSet, character-method (Depreciated), 45
getExpData (Depreciated), 45
getExpData, eSet, character-method (Depreciated), 45
isCurrent, 12, 18, 21, 23, 24, 29, 35, 52
isCurrent, ANY, ANY-method (isCurrent), 52
isCurrent, MIAME, missing-method (MIAME), 22
isCurrent, Versioned, character-method (Versioned), 31
isCurrent, Versioned, missing-method (Versioned), 31
isUnique, 53
isVersioned, 12, 18, 21, 23, 24, 29, 35, 54
isVersioned, ANY-method (isVersioned), 54
isVersioned, character-method (isVersioned), 54
isVersioned, Versioned-method (Versioned), 31
hybridizations (abstract), 52
hybridizations, MIAME-method (MIAME), 22
initialize, eSet-method (eSet), 16
initialize, ExpressionSet-method (ExpressionSet), 19
initialize, exprSet-method (Depreciated), 45
initialize, MultiSet-method (MultiSet), 24
initialize, NChannelSet-method (NChannelSet-class), 26
initialize, phenoData-method (Depreciated), 45
initialize, SnpSet-method (SnpSet), 28
initialize, Versioned-method (Versioned), 31
initialize, Versions-method (Versions), 34
initialize, VersionsNull-method (VersionsNull), 33
isCurrent, 12, 18, 21, 23, 24, 29, 35, 52
isCurrent, ANY, ANY-method (isCurrent), 52
isCurrent, MIAME, missing-method (MIAME), 22
isCurrent, Versioned, character-method (Versioned), 31
isCurrent, Versioned, missing-method (Versioned), 31
isUnique, 53
isVersioned, 12, 18, 21, 23, 24, 29, 35, 54
isVersioned, ANY-method (isVersioned), 54
isVersioned, character-method (isVersioned), 54
isVersioned, Versioned-method (Versioned), 31
l2e (copyEnv), 38
lcPrefix (lcSuffix), 55
lcPrefixC (lcSuffix), 55
lcSuffix, 55
length, container-method (container), 15
listLen, 56
listOrEnv (eSet), 16
listOrEnv-class (abstract), 52
locked (abstract), 52
locked, container-method (container), 15
makeDataPackage, 21, 57
makeDataPackage, ANY-method (makeDataPackage), 57
makeDataPackage,ExpressionSet-method  
(ExpressionSet), 19
makeProbePackage, 41, 42
matchpt, 58
matrix, 4, 69
menu, 63
merge, 36
MIAME, 6, 14, 22, 26, 36, 68
MIAME-class, 1, 18, 45, 46
MIAME-class (MIAME), 22
mkScalar (ScalarObject-class), 71
multiassign, 59
MultiSet, 6, 24
MultiSet-class, 18
MultiSet-class (MultiSet), 24
NA, 69
NChannelSet (NChannelSet-class), 26
NChannelSet-class, 26
nchar, 55
ncol, AnnotatedDataFrame-method  
(AnnotatedDataFrame), 11
ncol, eSet-method (eSet), 16
new.env, 3
normControls (abstract), 52
normControls, MIAME-method  
(MIAME), 22
note, 60
notes, 61
notes, eSet-method (eSet), 16
notes<-(notes), 61
notes<-, eSet, ANY-method (eSet), 16
notes<-, MIAME, character-method  
(MIAME), 22
notes<-, MIAME, list-method  
(MIAME), 22
numeric, 69
openPDF, 6, 61, 63
openVignette, 6, 52, 62
otherInfo (abstract), 52
otherInfo, MIAME-method (MIAME), 22
package.description, 63
package.version, 6, 63
package_version, 31, 34
pData, 11
pData (phenoData), 64
pData, AnnotatedDataFrame-method  
(AnnotatedDataFrame), 11
pData, eSet-method (eSet), 16
pData<-(phenoData), 64
pData<-, AnnotatedDataFrame, data.frame-method  
(AnnotatedDataFrame), 11
pData<-, eSet, data.frame-method  
(eSet), 16
phenoData, 64
phenoData, eSet-method (eSet), 16
phenoData-class (Deprecated), 45
phenoData<-(phenoData), 64
phenoData<-, eSet, AnnotatedDataFrame-method  
(eSet), 16
preproc (MIAME), 22
preproc, eSet-method (eSet), 16
preproc, MIAME-method (MIAME), 22
preproc<-(MIAME), 22
preproc<-, eSet-method (eSet), 16
preproc<-, MIAME-method (MIAME), 22
pubMedIds (abstract), 1
pubMedIds, eSet-method (eSet), 16
pubMedIds, MIAME-method (MIAME), 22
pubMedIds<-(abstract), 1
pubMedIds<-, eSet, character-method  
(eSet), 16
pubMedIds<-, MIAME, ANY-method  
(MIAME), 22
read.AnnotatedDataFrame, 13, 64, 66, 67
read.exprSet (Deprecated), 45
read.MIAME, 23, 66, 67, 67
read.pD (Deprecated), 45
read.phenoData (Deprecated), 45
read.table, 65–67
readExpressionSet, 66
readLines, 39, 66, 67
reporter, 43
reporterNames (Deprecated), 45
reporterNames, eSet-method  
(Deprecated), 45
reporterNames<-(Deprecated), 45
reporterNames<-, eSet, character-method  
(Deprecated), 45
reverseSplit, 68
rowMax (rowQ), 70
rowMedians, 69, 71
rowMedians, ExpressionSet-method  
(rowMedians), 69
rowMedians, matrix-method  
(rowMedians), 69
rowMin (rowQ), 70
rowQ, 70, 70
rowQ, ExpressionSet, numeric-method (rowQ), 70
rowQ, matrix, numeric-method (rowQ), 70
sample.eSet (Deprecated), 45
sample.ExpressionSet, 6
sample.ExpressionSet (data: sample.ExpressionSet), 44
sample.exprSet (Deprecated), 45
sample.MultiSet (data: sample.MultiSet), 45
sampleNames (featureNames), 51
sampleNames, AnnotatedDataFrame-method (AnnotatedDataFrame), 11
sampleNames, AssayData-method (AssayData-class), 13
sampleNames, eSet-method (eSet), 16
sampleNames, NChannelSet-method (NChannelSet-class), 26
sampleNames<- (featureNames), 51
sampleNames<-, AnnotatedDataFrame, ANY-method (AnnotatedDataFrame), 11
sampleNames<-, AssayData, ANY-method (AssayData-class), 13
sampleNames<-, AssayData, list-method (AssayData-class), 13
sampleNames<-, eSet, ANY-method (eSet), 16
sampleNames<-, NChannelSet, list-method (NChannelSet-class), 26
samples (MIAME), 22
samples, MIAME-method (MIAME), 22
sapply, 57
ScalarCharacter-class (ScalarObject-class), 71
ScalarInteger-class (ScalarObject-class), 71
ScalarLogical-class (ScalarObject-class), 71
ScalarNumeric-class (ScalarObject-class), 71
ScalarObject-class (ScalarObject-class), 71
scan, 68
se.exprs (exprs), 49
se.exprs<- (exprs), 49
search, 51, 59
seD (data: geneData), 43
selectChannels, 72
selectChannels, NChannelSet, character-method (NChannelSet-class), 26
selectSome, 72
show, 73
show, AnnotatedDataFrame-method (AnnotatedDataFrame), 11
show, container-method (container), 15
show, eSet-method (eSet), 16
show, MIAME-method (MIAME), 22
show, ScalarCharacter-method (ScalarObject-class), 71
show, ScalarObject-method (ScalarObject-class), 71
show, Versioned-method (Versioned), 31
show, Versions-method (Versions), 34
show, VersionsNull-method (VersionsNull), 33
SnpSet, 28
SnpSet-class, 4, 6, 16, 18, 50, 51, 61, 64, 73, 78
SnpSet-class (SnpSet), 28
split, 69
stop, 39, 60
storageMode, 73
storageMode, AssayData-method (AssayData-class), 13
storageMode, eSet-method (eSet), 16
storageMode<- (storageMode), 73
storageMode<-, AssayData, character-method (AssayData-class), 13
storageMode<-, eSet, character-method (eSet), 16
strbreak, 74
strwrap, 74
subListExtract, 74
substring, 74
SW (eSet), 16
sys.frame, 59
testBioCConnection, 75
tkMIAME, 68
TRUE, 5, 69
unique, 54
updateObject, 12, 18, 21, 23, 24, 26, 29, 76
updateObject, AnnotatedDataFrame-method (AnnotatedDataFrame), 11
updateObject, ANY-method
updateObject, environment-method (updateObject), 76
updateObject,eSet-method (eSet), 16
updateObject,ExpressionSet-method (ExpressionSet), 19
updateObject,list-method (updateObject), 76
updateObject,MIAME-method (MIAME), 22
updateObject,Versions-method (Versions), 34
updateObjectFromSlots, 76
updateObjectFromSlots (updateObject), 76
updateObjectTo, 18
updateObjectTo (updateObject), 76
updateObjectTo,ANY,ANY-method (updateObject), 76
updateObjectTo,eSet,eSet-method (eSet), 16
updateOldESet, 18, 24, 78
updateOldMiame (Deprecated), 45
userQuery, 79
validMsg, 79
validObject, 76
varLabels (phenoData), 64
varLabels,AnnotatedDataFrame-method (AnnotatedDataFrame), 11
varLabels,eSet-method (eSet), 16
varLabels<-(phenoData), 64
varLabels<-,AnnotatedDataFrame-method (AnnotatedDataFrame), 11
varLabels<-,eSet-method (eSet), 16
varMetadata, 11
varMetadata (phenoData), 64
varMetadata,AnnotatedDataFrame-method (AnnotatedDataFrame), 11
varMetadata,eSet-method (eSet), 16
varMetadata<-(phenoData), 64
varMetadata<-,AnnotatedDataFrame,data.frame-method (AnnotatedDataFrame), 11
varMetadata<-,eSet,data.frame-method (eSet), 16
vector, 5, 69
Versioned, 6, 27, 31
Versioned-class, 30, 32, 34, 52, 54
Versioned-class, 54
Versioned-class (Versioned), 31
VersionedBiobase, 6, 27, 30
VersionedBiobase-class (VersionedBiobase), 30
Versions, 26, 34
Versions-class, 31–33, 53, 54, 77