GSEABase

April 20, 2011

`CollectionType-class`
Class "CollectionType"

Description

These classes provides a way to tag the origin of a `GeneSet`. Collection types can be used in manipulating (e.g., selecting) sets, and can contain information specific to particular sets (e.g., `category` and `subcategory` classifications of `BroadCollection`.)

Objects from the Class

The following classes can tag gene sets; `GO, KEGG, Chr, Chrloc, OMIM, and PMID` collections can be derived from chip or organism 'annotation' packages.

NullCollection No formal collection information available.

BroadCollection Derived from, or destined to be, Broad XML. Usually created and written `getBroadSets`, `toBroadXML`.

ComputedCollection A computationally created collection, e.g., by performing logic operations on gene sets.

ExpressionSetCollection Derived from `ExpressionSet`. Usually created during a call to `GeneSet` or `GeneColorSet`.

GOCollection Collection derived using Gene Ontology (GO) terms.

OBOCollection Collection derived from `GOCollection`, specifically from files described by the OBO file format. See `OBOCollection`

KEGGCollection Collection derived using KEGG terms.

ChrCollection Collection derived using chromsome locations

ChrlocCollection Collection derived using chromosome starting posistions

MapCollection Collection derived from cytogenic bands.

OMIMCollection Collection derived from identifiers in the Online Inheritance in Man.

PMIDCollection Collection derived from PMID identifiers.

PfamCollection Collection derived from Pfam identifiers.

PrositeCollection Collection derived from Prosite identifiers.

Objects are instantiated with calls to `CollectionType` constructors, with slot names as possible arguments.

Slots

`CollectionType` classes (`Null, ComputedCollection, ExpressionSet`) have the slot:

`type`**:** Object of class `"ScalarCharacter"` containing the character string representation of this `CollectionType`.

`CollectionIdType` classes (`KEGG, OMIM, PMID, Chr, Chrloc, Map, GO`) extend the `CollectionType` and have the additional slot:

`ids`**:** Object of class `"character"` containing a vector of character string representations of corresponding identifiers, e.g., 'KEGG' or 'GO' terms.

`GOCollection` extends `CollectionIdType` and has the additional slot:

`evidenceCode`**:** Object of class `"character"`, containing GO evidence codes used to construct the gene set.

ontology Object of class `"character"` vector of GO ontology terms used to filter GO terms in the GO Collection.

The values of `evidenceCode` are

Experimental Evidence Codes **EXP** Inferred from Experiment

IDA Inferred from Direct Assay

IPI Inferred from Physical Interaction

IMP Inferred from Mutant Phenotype

IGI Inferred from Genetic Interaction

IEP Inferred from Expression Pattern

Computational Analysis Evidence Codes **ISS** Inferred from Sequence or Structural Similarity

ISO Inferred from Sequence Orthology

ISA Inferred from Sequence Alignment

ISM Inferred from Sequence Model

IGC Inferred from Genomic Context

RCA inferred from Reviewed Computational Analysis

Author Statement Evidence Codes **TAS** Traceable Author Statement

NAS Non-traceable Author Statement

Curator Statement Evidence Codes **IC** Inferred by Curator

ND No biological Data available

Automatically-assigned Evidence Codes **IEA** Inferred from Electronic Annotation

`OBOCollection` extends `GOCollection`; see `OBOCollection`.

`BroadCollection` has slots:

`category`**:** Object of class `"ScalarCharacter"` containing terms from the Broad list of categories, or `NA`

`subCategory`**:** Object of class `"ScalarCharacter"` containing Broad sub-categories, or `NA`

Methods

`CollectionType` classes have methods:

collectionType<- `signature(object = "GeneSet", value = "CollectionType")`: Replace the `CollectionType`

collectionType `signature(object = "CollectionType")`: Retrieve the collection type.

|, &, intersect, union, setdiff `signature(e1="CollectionType", e2="CollectionType")`: return `e1` when `class(e1)` and `class(e2)` are the same, or `ComputedCollection` when different.

show `signature(object = "CollectionType")`: display the collection type.

`CollectionIdType` classes inherit `CollectionType` methods, and have in addition:

ids `signature(object="CollectionIdType")`: Retrieve the identifiers of the collection type.

[`signature(object="CollectionIdType", i="missing", j="missing", ..., ids=ids(object))`: return a subset of `object` containing only ids in `ids`

|, &, intersect, union, setdiff `signature(e1="CollectionIdType", e2="CollectionIdType")`: always return `ComputedCollection`.

`GOCollection` inherits `CollectionIdType` methods, and has in addition:

evidenceCode Retrieve the evidence codes of the GO collection.

ontology Retrieve the ontology terms of the GO collection.

[`signature(object="CollectionIdType", i="missing", j="missing", ..., evidenceCode=evidenceCode(object), ontology=ontology(object))`: return a subset of `object` containing only evidence and ontology codes in `evidenceCode`, `ontology`. This method passes arguments ... to `[,CollectionIdType` methods.

`BroadCollection` has methods:

bcCategory Retrieve the `category` of the Broad collection.

bcSubCategory Retrieve the sub-category of the Broad collection.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

See Also

`CollectionType` consturctors; `getBroadSets` for importing collections from the Broad (and sources).

Examples

```
names(getClass("CollectionType")@subclasses)

## Create a CollectionType and ask for its type
collectionType(ExpressionSetCollection())

## Read two GeneSets from a Broad XML file into a list, verify that
## they are both BroadCollection's. Category / subcategory information
## is unique to Broad collections.
```

```
fl <- system.file("extdata", "Broad.xml", package="GSEABase")
sets <- getBroadSets(fl)
sapply(sets, collectionType)

## ExpressionSets are tagged with ExpressionSetCollection; there is no
## 'category' information.
data(sample.ExpressionSet)
gs <- GeneSet(sample.ExpressionSet[100:109],
              setName="sample.GeneSet", setIdentifier="123")
collectionType(gs)

## GOCollections are created by reference to GO terms and evidenceCodes
GOCollection("GO:0005488")
## requires library(GO); EntrezIdentifers automatically created
## Not run:
GeneSet(GOCollection(c("GO:0005488", "GO:0019825"),
                     evidenceCode="IDA"))

## End(Not run)
```

`CollectionType` *Collection Type Class Constructors*

Description

These functions construct collection types. Collection types can be used in manipulating (e.g., selecting) sets, and can contain information specific to particular sets (e.g., 'category' and 'subcategory' classifications of 'BroadCollection'.)

Usage

```
NullCollection(...)
ComputedCollection(...)
ExpressionSetCollection(...)
ChrCollection(ids,...)
ChrlocCollection(ids,...)
KEGGCollection(ids,...)
MapCollection(ids,...)
OMIMCollection(ids,...)
PMIDCollection(ids,...)
PfamCollection(ids, ...)
PrositeCollection(ids, ...)
GOCollection(ids=character(0), evidenceCode="ANY", ontology="ANY", ..., err=FALS
OBOCollection(ids, evidenceCode="ANY", ontology="ANY", ...)
BroadCollection(category, subCategory=NA, ...)
```

Arguments

`category`	(Required) Broad category, one of "c1" (postitional), "c2" (curated), "c3" (motif), "c4" (computational), "c5" (GO).
`subCategory`	(Optional) Sub-category; no controlled vocabulary.
`ids`	(Optional) Character vector of identifiers (e.g., GO, KEGG, or PMID terms).

`evidenceCode` (Optional) Character vector of GO evidence codes to be included, or "ANY" (any identifier; the default). Evidence is a property of particular genes, rather than of the ontology, so evidenceCode is a convenient way of specifying how users of a GOCollection might restrict derived objects (as in done during create of a gene set from an expression set).

`ontology` (Optional) Character vector of GO ontology terms to be included, or "ANY" (any identifier; the default). Unlike evidence code, ontology membership is enforced when GOCollection gene sets are constructed.

`err` (Optional) logical scalar indicating whether non-existent GO terms signal an error (`TRUE`), or are silently ignored (`FALSE`).

`...` Additional arguments, usually none but see specific `linkS4class{CollectionType}` classes for possibilities.

Value

An object of the same class as the function name, initialized as appropriate for the collection.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

See Also

`CollectionType`,

Examples

```
NullCollection()

## NullCollection when no collection type specified
collectionType(GeneSet())
collectionType(GeneSet(collectionType=GOCollection()))

## fl could be a url
fl <- system.file("extdata", "Broad.xml", package="GSEABase")
gs1 <- getBroadSets(fl)[[1]]
collectionType(gs1) # BroadCollection

## new BroadCollection, with different category
bc <- BroadCollection(category="c2")
## change collectionType of gs2
gs2 <- gs1
collectionType(gs2) <- NullCollection()

## OBOCollection
fl <- system.file("extdata", "goslim_plant.obo", package="GSEABase")
getOBOCollection(fl, evidenceCode="TAS") # returns OBOCollection
OBOCollection(c("GO:0008967", "GO:0015119", "GO:0030372", "GO:0002732",
                "GO:0048090"))
```

details-methods *Methods for Displaying Detailed GeneSet Information*

Description

This generic and methods supplement show, providing more detail on object contents.

Methods

Defined methods include:

These methods display information about setIdentifier, description, organism, pubMedIds, urls, contributor, setVersion, and creationDate.

signature(object = "GeneSet"), signature(object = "GeneColorSet")

GeneColorSet-class *Class "GeneColorSet"*

Description

A GeneColorSet extends GeneSet to allow genes to be 'colored'. Coloring means that for a particular phenotype, each gene has a color (e.g., expression levels "up", "down", or "unchanged") and a phenotypic consequence (e.g., the phenotype is "enhanced" or "reduced").

All operations on a GeneSet can be applied to a GeneColorSet; coloring can also be accessed.

Objects from the Class

Construct a GeneColorSet with a GeneColorSet method. These methods are identical to those for GeneSet, except they require an additional phenotype argument to specify the phenotype to which the genetic and phenotypic coloring apply. A GeneColorSet can be constructed from a GeneSet with GeneColorSet(<GeneSet>, phenotype="<phenotype>").

Slots

A GeneColorSet inherits all slots from GeneSet, and gains the following slots:

phenotype: Object of class "ScalarCharacter" describing the phenotype for which this gene set is colored.

geneColor: Object of class "factor" describing the coloring of each gene in the set. The lengths of geneColor and gene must be equal.

phenotypeColor: Object of class "factor" describing the phenotypic coloring of each gene in the set. The lengths of phenotypeColor and gene must be equal.

Extends

Class "GeneSet", directly.

Methods

Methods unique to `GeneColorSet` include:

coloring `signature(object = "GeneColorSet")`: retrieve coloring as a `data.frame`. The row names of the data frame are the gene names; the columns are `geneColor` and `phenotypeColor`.

coloring<- `signature(object = "GeneColorSet", value = "data.frame")`: use a `data frame` to assign coloring information. The `data.frame` must have the same number of rows as the `GeneColorSet` has genes (though see the examples below for flexible ways to alter coloring of a subset of genes). Row names of the `data.frame` correspond to gene names. The data frame has two columns, named `geneColor` and `phenotypeColor`. These must be of class `factor`.

A typical use of `coloring<-` is to simultaneous extract, subset, and reassign the current coloring, e.g., `coloring(<GeneColorSet>)[1:5,"geneColor"] <- "up"`; see the examples below.

geneColor<- `signature(object = "GeneColorSet", value = "factor")`: assign gene colors.

geneColor `signature(object = "GeneColorSet")`: retrieve gene colors as a `factor`.

phenotypeColor<- `signature(object = "GeneColorSet", value = "factor")`: assign phenotype colors.

phenotypeColor `signature(object = "GeneColorSet")`: retrieve phenotype colors as a `factor`.

phenotype<- `signature(object = "GeneColorSet", value = "character")`: assign the phenotype from a single-element character vector.

phenotype `signature(object = "GeneColorSet")`: retrieve the phenotype as a single-element `character`.

`GeneColorSet` inherits all methods from class `GeneSet`. Methods with different behavior include

[`signature(x = "GeneSet", i="character")` `signature(x = "GeneSet", i="numeric")`: subset the gene set by index (`i="numeric"`) or gene value (`i="character"`). Genes are re-ordered as required. `geneColor` and `phenotypeColor` are subset as appropriate.

[[`signature(x = "GeneSet")`: select a single gene from the gene set, returning a named character vector of `gene, geneColor, phenotypeColor`. Exact matches only.

\$ `signature(x = "GeneSet")`: select a single gene from the gene set, returning a named character vector of `gene, geneColor, phenotypeColor`. Provides partial matching into the list of genes.

mapIdentifiers `signature(x="GeneColorSet", to="*", from="*")`: checks that gene- and phenotype colors are consistent for mapped identifiers, e.g., that two `AnnotationIdentifiers` mapping to the same `SymbolIdentifier` are colored the same.

Logical (set) operations `&, |, setdiff` warn if the `phenotype` `geneColor`, or `phenotypeColor` differs between sets; this implies coercion of factor levels, and the consequences should be carefully considered.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

See Also

GeneSet.

Examples

```
## Create a GeneColorSet from an ExpressionSet
data(sample.ExpressionSet)
gcs1 <- GeneColorSet(sample.ExpressionSet[100:109],
                     phenotype="imaginary")
gcs1
## or with color...
gcs2 <- GeneColorSet(sample.ExpressionSet[100:109],
                     phenotype="imaginary",
                     geneColor=factor(
                       rep(c("up", "down", "unchanged"),
                           length.out=10)),
                     phenotypeColor=factor(
                       rep(c("enhanced", "reduced"),
                           length.out=10)))
coloring(gcs2)

## recode geneColor of genes 1 and 4
coloring(gcs2)[c(1,4),"geneColor"] <- "down"
coloring(gcs2)
## reset, this time by gene name
coloring(gcs2)[c("31339_at", "31342_at"),"geneColor"] <- c("up", "up")
## usual 'factor' errors and warning apply:
coloring(gcs2)[c("31339_at", "31342_at"),"geneColor"] <- c("UP", "up")

gcs2[["31342_at"]]
try(gcs2[["31342_"]]) # no partial matching
gcs2$"31342" # 1 partial match ok
```

GeneColorSet *Methods to Construct "GeneColorSet" Instances*

Description

GeneColorSet is a generic for constructing gene color sets (i.e., gene sets with "coloring" to indicate how features of genes and phenotypes are associated).

Methods

Available methods are the same as those for GeneSet, but a GeneColorSet requires an additional phenotype argument to identify the phenotype that is being colored. See documentation for GeneColorSet for examples.

An additional method is:

signature(type = "GeneSet", phenotype="character") This method constructs a 'color' gene set from an uncolored gene set.

See Also

GeneColorSet-class

GeneIdentifierType-class
Class "GeneIdentifierType"

Description

This class provides a way to tag the meaning of gene symbols in a GeneSet. For instance, a GeneSet with gene names derived from a Bioconductor annotation package (e.g., via ExpressionSet) initially have a GeneIdentifierType of AnnotationIdentifier.

Objects from the Class

The following classes are available, and derive from tables in 'annotation' packages

NullIdentifier No formal information about what gene identifiers represent.

AnnotationIdentifier Gene identifiers are Affymetrix chip-specific probe identifier, as represented in Bioconductor annotation packages.

EntrezIdentifier 'Entrez' identifiers.

EnzymeIdentifier 'EC' identifiers.

ENSEMBLIdentifier 'ENSEMBL' identifiers.

GenenameIdentifier Curated and ad hoc descriptive gene names.

RefseqIdentifier 'Prosite' identifiers.

SymbolIdentifier 'Symbol' identifiers.

UnigeneIdentifier 'Unigene' identifiers.

GeneIdentifierType A virtual Class: No objects may be created from it; all classes listed above are subclasses of GeneIdentifierType.

Slots

All GeneIdentifierType classes have the following slots:

type Object of class "ScalarCharacter" containing the character string representation of this GeneIdentifierType.

annotation Object of class "ScalarCharacter" containing the name of the annotation package from which the identifiers (probe identifiers) are derived.

Methods

GeneIdentifierType classes are used in:

GeneSet signature(type = "GeneIdentifierType"): Create a new GeneSet using identifiers of GeneIdentifierType.

GeneColorSet signature(type = "GeneIdentifierType"): Create a new GeneColorSet using identifiers of GeneIdentifierType.

annotation signature(object = "GeneIdentifierType"): extract the name of the annotation package as a character string.

annotation<- signature(object = "GeneIdentifierType", value = "character"): assign the name of the annotation package as a character string.

geneIdType `signature(object = "GeneIdentifierType")`: return a character string representation of the type of this `object`.

geneIdType<- `signature(object = "GeneSet", verbose=FALSE, value = "GeneIdentifierT` Changes the `GeneIdentifierType` of `object` to `value`, attempting to convert symbols in the process. This method calls `mapIdentifiers(what=object, to=value, from=geneIdType(what), verbose=verbose)`.

mapIdentifiers See `mapIdentifiers`.

show `signature(object = "GeneIdentifierType")`: display this object.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

See Also

The example below lists `GeneIdentifierType` classes defined in this package; See the help pages of these classes for specific information.

Examples

```
names(getClass("GeneIdentifierType")@subclasses)

# create an AnnotationIdentifier, and ask it's type
geneIdType(AnnotationIdentifier(annotation="hgu95av2"))

# Construct a GeneSet from an ExpressionSet, using the 'annotation'
# field of ExpressionSet to recognize the genes as AnnotationType
data(sample.ExpressionSet)
gs <- GeneSet(sample.ExpressionSet[100:109],
              setName="sample.GeneSet", setIdentifier="123")
geneIdType(gs) # AnnotationIdentifier

## Read a Broad set from the system (or a url), and discover their
## GeneIdentifierType
fl <- system.file("extdata", "Broad.xml", package="GSEABase")
bsets <- getBroadSets(fl)
sapply(bsets, geneIdType)

## try to combine gene sets with different set types
try(gs & sets[[1]])

## Not run:
## Use the annotation package associated with the original
## ExpressionSet to map to EntrezIdentifier() ...
geneIdType(gs) <- EntrezIdentifier()
## ...and try again
gs & bsets[[1]]

## Another way to change annotation to Entrez (or other) ids
probeIds <- featureNames(sample.ExpressionSet)[100:109]
geneIds <- getEG(probeIds, "hgu95av2")
GeneSet(EntrezIdentifier(),
        setName="sample.GeneSet2", setIdentifier="101",
        geneIds=geneIds)
```

```
## End(Not run)

## Create a new identifier
setClass("FooIdentifier",
         contains="GeneIdentifierType",
         prototype=prototype(
           type=new("ScalarCharacter", "Foo")))
## Create a constructor (optional)
FooIdentifier <- function() new("FooIdentifier")
geneIdType(FooIdentifier())

## tidy up
removeClass("FooIdentifier")
```

`GeneIdentifierType` *Gene Identifier Class Constructors*

Description

Gene identifier classes and functions are used to indicate what the list of genes in a gene set represents (e.g., Entrez gene identifiers are tagged with `EntrezIdentifier()`, Bioconductor annotations with `AnnotationIdentifier()`).

Usage

```
NullIdentifier(annotation, ...)
EnzymeIdentifier(annotation, ...)
ENSEMBLIdentifier(annotation, ...)
GenenameIdentifier(annotation,...)
RefseqIdentifier(annotation,...)
SymbolIdentifier(annotation,...)
UnigeneIdentifier(annotation,...)
EntrezIdentifier(annotation,...)
AnnotationIdentifier(annotation, ...)
```

Arguments

`annotation`	An optional character string identifying the Bioconductor package from which the annotations are drawn, e.g., 'hgu95av2', 'org.Hs.eg.db'.
`...`	Additional arguments, usually none.

Value

An object of the same class as the function name, initialized as appropriate for the identifier.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

See Also

`GeneIdentifierType`-class for a description of the classes and methods using these objects.

Examples

```
NullIdentifier()

data(sample.ExpressionSet)
gs1 <- GeneSet(sample.ExpressionSet[100:109],
               setName="sample1", setIdentifier="100")
geneIdType(gs1) # AnnotationIdentifier

geneIds <- featureNames(sample.ExpressionSet)[100:109]
gs2 <- GeneSet(geneIds=geneIds,
               setName="sample1", setIdentifier="101")
geneIdType(gs2) # NullIdentifier, since no info about genes provided

## Convert...
ai <- AnnotationIdentifier(annotation(sample.ExpressionSet))
geneIdType(gs2) <- ai
geneIdType(gs2)
## ...or provide more explicit construction
gs3 <- GeneSet(geneIds=geneIds, type=ai,
               setName="sample1", setIdentifier="102")
```

GeneSet-class *Class "GeneSet"*

Description

A `GeneSet` contains a set of gene identifiers. Each gene set has a `geneIdType`, indicating how the gene identifiers should be interpreted (e.g., as Entrez identifiers), and a `collectionType`, indicating the origin of the gene set (perhaps including additional information about the set, as in the `BroadCollection` type).

Conversion between identifiers, subsetting, and logical (set) operations can be performed. Relationships between genes and phenotype in a `GeneSet` can be summarized using `coloring` to create a `GeneColorSet`. A `GeneSet` can be exported to XML with `toBroadXML`.

Objects from the Class

Construct a `GeneSet` with a `GeneSet` method (e.g., from a character vector of gene names, or an `ExpressionSet`), or from gene sets stored as XML (locally or on the internet; see `getBroadSets`)

Slots

`setName`: Object of class `"ScalarCharacter"` containing a short name (single word is best) to identify the set.

`setIdentifier`: Object of class `"ScalarCharacter"` containing a (unique) identifier for the set.

`geneIdType`: Object of class `"GeneIdentifierType"` containing information about how the gene identifiers are encoded. See `GeneIdentifierType` and related classes.

`geneIds`: Object of class `"character"` containing the gene symbols.

`collectionType`: Object of class `"CollectionType"` containing information about how the geneIds were collected, including perhaps additional information unique to the collection methodology. See `CollectionType` and related classes.

`shortDescription`: Object of class `"ScalarCharacter"` representing short description (1 line) of the gene set.

`longDescription`: Object of class `"ScalarCharacter"` providing a longer description (e.g., like an abstract) of the gene set.

`organism`: Object of class `"ScalarCharacter"` represents the organism the gene set is derived from.

`pubMedIds`: Object of class `"character"` containing PubMed ids related to the gene set.

`urls`: Object of class `"character"` containing urls used to construct or manipulate the gene set.

`contributor`: Object of class `"character"` identifying who created the gene set.

`version`: Object of class `"Versions"` a version number, manually curated (i.e., by the `contributor`) to provide a consistent way of tracking a gene set.

`creationDate`: Object of class `"character"` containing the character string representation of the date on which the gene set was created.

Methods

Gene set construction:

GeneSet See `GeneSet` methods and `getBroadSets` for convenient construction.

Slot access (e.g., `setName`) and retrieve (e.g., `setName<-`) :

collectionType<- `signature(object = "GeneSet", value = "CollectionType")`

collectionType `signature(object = "GeneSet")`

contributor<- `signature(object = "GeneSet", value = "character")`

contributor `signature(object = "GeneSet")`

creationDate<- `signature(object = "GeneSet", value = "character")`

creationDate `signature(object = "GeneSet")`

description<- `signature(object = "GeneSet", value = "character")`

description `signature(object = "GeneSet")`

geneIds<- `signature(object = "GeneSet", value = "character")`

geneIds `signature(object = "GeneSet")`

longDescription<- `signature(object = "GeneSet", value = "character")`

longDescription `signature(object = "GeneSet")`

organism<- `signature(object = "GeneSet", value = "character")`

organism `signature(object = "GeneSet")`

pubMedIds<- `signature(object = "GeneSet", value = "character")`

pubMedIds `signature(object = "GeneSet")`

setdiff `signature(x = "GeneSet", y = "GeneSet")`

setIdentifier<- `signature(object = "GeneSet", value = "character")`

setIdentifier `signature(object = "GeneSet")`

setName<- `signature(object = "GeneSet", value = "character")`

setName `signature(object = "GeneSet")`

geneIdType<- `signature(object = "GeneSet", verbose=FALSE, value = "character"), signature(object = "GeneSet", verbose=FALSE, value = "GeneIdentifierType")`: These method attempt to coerce geneIds from the current type to the type named by `value`. Successful coercion requires an appropriate method for `mapIdentifiers`.

geneIdType `signature(object = "GeneSet")`

setVersion<- `signature(object = "GeneSet", value = "Versions")`

setVersion `signature(object = "GeneSet")`

urls<- `signature(object = "GeneSet", value = "character")`

urls `signature(object = "GeneSet")`

Logical and subsetting operations:

union `signature(x = "GeneSet", y = "GeneSet")`: ...

| `signature(e1 = "GeneSet", e2 = "GeneSet")`: calculate the logical 'or' (union) of two gene sets. The sets must contain elements of the same `geneIdType`.

| `signature(e1 = "GeneSet", e2 = "character"), signature(e1 = "character", e2 = "GeneSet")`: calculate the logical 'or' (union) of a gene set and a character vector, i.e., add the geneIds named in the character vector to the gene set.

intersect `signature(x = "GeneSet", y = "GeneSet")`:

& `signature(e1 = "GeneSet", e2 = "GeneSet")`: calculate the logical 'and' (intersection) of two gene sets.

& `signature(e1 = "GeneSet", e2 = "character"), signature(e1 = "character", e2 = "GeneSet")`: calculate the logical 'and' (intersection) of a gene set and a character vector, creating a new gene set containing only those genes named in the character vector.

setdiff `signature(x = "GeneSet", y = "GeneSet"), signature(x = "GeneSet", y = "character"), signature(x = "character", y = "GeneSet")`: calculate the logical set difference betwen two gene sets, or betwen a gene set and a character vector.

[`signature(x = "GeneSet", i="character") signature(x = "GeneSet", i="numeric")`: subset the gene set by index (`i="numeric"`) or value (`i="character"`). Genes are reordered as required

[`signature(x = "ExpressionSet", i = "GeneSet")`: subset the expression set, using genes in the gene set to select features. Genes in the gene set are coerced to appropriate annotation type if necessary (by consulting the `annotation` slot of the expression set, and using `geneIdType<-`).

[[`signature(x = "GeneSet")`: select a single gene from the gene set.

\$ `signature(x = "GeneSet")`: select a single gene from the gene set, allowing partial matching.

Useful additional methods include:

GeneColorSet `signature(type = "GeneSet")`: create a 'color' gene set from a `GeneSet`, containing information about phenotype. This method has a required argument `phenotype`, a character string describing the phenotype for which color is available. See `GeneColorSet`.

mapIdentifiers Use the code in the examples to list available methods. These convert genes from one `GeneIdentifierType` to another. See `mapIdentifiers` and specific methods in `GeneIdentifierType` for additional detail.

incidence Summarize shared membership in genes across gene sets. See `incidence-methods`.

toGmt Export to 'GMT' format file. See `toGmt`.

show `signature(object = "GeneSet")`: display a short summary of the gene set.

details `signature(object = "GeneSet")`: display additional information about the gene set. See `details`.

initialize `signature(.Object = "GeneSet")`: Used internally during gene set construction.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

See Also

`GeneColorSet` `CollectionType` `GeneIdentifierType`

Examples

```
## Empty gene set
GeneSet()
## Gene set from ExpressionSet
data(sample.ExpressionSet)
gs1 <- GeneSet(sample.ExpressionSet[100:109])
## GeneSet from Broad XML; 'fl' could be a url
fl <- system.file("extdata", "Broad.xml", package="GSEABase")
gs2 <- getBroadSets(fl)[[1]] # actually, a list of two gene sets
## GeneSet from list of geneIds
geneIds <- geneIds(gs2) # any character vector would do
gs3 <- GeneSet(geneIds=geneIds)
## unspecified set type, so...
is(geneIdType(gs3), "NullIdentifier") == TRUE
## update set type to match encoding of identifiers
geneIdType(gs2)
geneIdType(gs3) <- SymbolIdentifier()

## Convert between set types; this consults the 'annotation'
## information encoded in the 'AnnotationIdentifier' set type and the
## corresponding annotation package.
## Not run:
gs4 <- gs1
geneIdType(gs4) <- EntrezIdentifier()

## End(Not run)

## logical (set) operations
gs5 <- GeneSet(sample.ExpressionSet[100:109], setName="subset1")
gs6 <- GeneSet(sample.ExpressionSet[105:114], setName="subset2")
## intersection: 5 'genes'; note the set name '(subset1 & subset2)'
gs5 & gs6
## union: 15 'genes'; note the set name
gs5 | gs6
## an identity
gs7 <- gs5 | gs6
gs8 <- setdiff(gs5, gs6) | (gs5 & gs6) | setdiff(gs6, gs5)
identical(geneIds(gs7), geneIds(gs8))
identical(gs7, gs8) == FALSE # gs7 and gs8 setNames differ
```

```
## output
tmp <- tempfile()
toBroadXML(gs2, tmp)
noquote(readLines(tmp))
## must be BroadCollection() collectionType
try(toBroadXML(gs1))
gs9 <- gs1
collectionType(gs9) <- BroadCollection()
toBroadXML(gs9, tmp)
unlink(tmp)
toBroadXML(gs9) # no connection --> character vector
## list of geneIds --> vector of Broad GENESET XML
gs10 <- getBroadSets(fl) # two sets
entries <- sapply(gs10, function(x) toBroadXML(x)[[2]])

## list mapIdentifiers available for GeneSet
showMethods("mapIdentifiers", classes="GeneSet", inherit=FALSE)
```

GeneSetCollection-class *Class "GeneSetCollection"*

Description

a GeneSetCollection is a collection of related GeneSets. The collection can mix and match different types of gene sets. Members of the collection are refered to by the setNames of each gene set.

Objects from the Class

Construct a GeneSetCollection with a GeneSetCollection method, e.g., from a list of gene sets or with several gene sets provided as argument to the constructor. See examples below.

Slots

.Data: Object of class "list", containing the gene sets.

Extends

Class "list", from data part. Class "vector", by class "list", distance 2. Class "AssayData", by class "list", distance 2.

Methods

Gene set collection construction

GeneSetCollection See GeneSetCollection methods and getBroadSets for convenient construction methods.

Collection access (operations on lists, such as length, , lapply also work on GeneSetCollection).

geneIds signature(object = "GeneSetCollection"): return a list, with each member a character vector of gene identifiers from the gene set collection.

geneIds<- `signature(object="GeneSetCollection", value="list")`: assign character vectors in `value` to corresponding `geneIds` of `object`.

names `signature(x = "GeneSetCollection")`: return the `setName` of each gene set in the colloection.

Logical and subsetting operations

union `signature(x = "GeneSetCollection", y = "ANY"),signature(x = "ANY", y = "GeneSetCollection")`: ...

| `signature(e1 = "GeneSetCollection", e2 = "ANY"),signautre(e1 = "GeneSet", e2 = "GeneSetCollection"),signautre(e1 = "character", e2 = "GeneSetCollection"), signature(e1 = "ANY", e2 = "GeneSetCollection")`: calculate the logical 'or' (union) of all gene identifiers in an object over all members of the gene set collection.

intersect `signature(x = "GeneSetCollection", y = "ANY"),signature(x = "ANY", y = "GeneSetCollection")`: ...

& `signature(e1 = "GeneSetCollection", e2 = "ANY"),signautre(e1 = "character", e2 = "GeneSetCollection"),signautre(e1 = "GeneSet", e2 = "GeneSetCollection"), signature(e1 = "ANY", e2 = "GeneSetCollection")`: calculate the logical 'and' (intersection) of all gene identifiers in a gene set or character vector, over all members of the gene set collection.

setdiff `signature(x = "GeneSetCollection", y = "ANY")`: calculate the logical set difference betwen all gene sets in a collection and the gene identifiers of a gene set or character vector. A `setdiff` method must be available for `x="GeneSet"` and the type of `y`.

[<- `signature(x = "GeneSetCollection", i = "ANY", j = "ANY", value = "ANY"),signature(x = "GeneSetCollection", i = "ANY", j = "ANY", value = "GeneSet"),signature(x = "GeneSetCollection", i = "character", j = "ANY", value = "GeneSet")`: assign new sets to existing set members. To add entirely new sets, use a `GeneSetCollection` constructor.

[`signature(x = "GeneSetCollection", i = "logical"),signature(x = "GeneSetCollection", i = "numeric"),signature(x = "GeneSetCollection", i = "character")`: create a `GeneSetCollection` consisting of a subset of the current set. All indicies `i` must already be present in the set.

[[`signature(x = "GeneSetCollection", i = "character")`: Select a single gene set from the collection. Methods for `i="numeric"` are inherited from `list`.

[[<- `signature(x = "GeneSetCollection", i = "ANY", j = "ANY", value = "ANY"),signature(x = "GeneSetCollection", i = "numeric", j = "ANY", value = "GeneSet"),signature(x = "GeneSetCollection", i = "character", j = "ANY", value = "GeneSet")`: Replace a gene set in the collecton with another. `value = "ANY"` serves to stop invalid assignments.

Additional useful methods.

updateObject Objects created in previous versions of GSEABase may be incompatible with current object definitions. Usually this is singalled by an error suggesting that a slot is missing, and a recommnedation to use `updateObject`. Use `updateObject` to update a `GeneSetCollection` and all contained `GeneSets` to their current defintion.

mapIdentifiers Convert genes from one `GeneIdentifierType` to another. See `mapIdentifiers` and specific methods in `GeneIdentifierType` for additional detail.

incidence Summarize shared membership in genes across gene sets. See `incidence-methods`.

toGmt Export to 'GMT' format file. See `toGmt`.

show `signature(object="GeneSetCollection")`: provide a compact representation of `object`.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

See Also

GeneSet, GeneColorSet.

Examples

```
gs1 <- GeneSet(setName="set1", setIdentifier="101")
gs2 <- GeneSet(setName="set2", setIdentifier="102")

## construct from indivdiual elements...
gsc <- GeneSetCollection(gs1, gs2)
## or from a list
gsc <- GeneSetCollection(list(gs1, gs2))

## 'names' are the setNames
names(gsc)

## a collection of a single gene set
gsc["set1"]
## a gene set
gsc[["set1"]]

## set names must be unique
try(GeneSetCollection(gs1, gs1))
try(gsc[c("set1", "set1")])
```

GeneSetCollection-methods

Methods to construct GeneSetCollection instances

Description

Use GeneSetCollection to construct a collection of gene sets from GeneSet arguments, or a list of GeneSets.

Usage

```
GeneSetCollection(object, ..., idType, setType)
```

Arguments

object	An argument determining how the gene set collection will be created, as described in the methods section.
...	Additional arugments for gene set collection construction, as described below.
idType	An argument of class GeneIdentifierType, used to indicate how the geneIds will be represented.
setType	An argument of class CollectionType, used to indicate how the collection is created.

Methods

signature(object = "GeneSet",idType="missing", setType="missing")
Construct a gene set collection from one or more GeneSet arugments.

signature(object = "list", idType="missing", setType="missing") Construct a gene set collection from a list of GeneSets.

signature(object="missing", idType="AnnotationIdentifier", setType="CollectionTy
Construct a gene set collection of CollectionType entities (e.g., pathways for KEGGCollection, protein families for PfamCollection) implied by the map found in annotation(idType). If setType is a CollectionIdType and length(ids(setType))>0, the gene set collection is filtered to contain only those sets implied by the ids.

signature(object="character", idType="AnnotationIdentifier", setType="Collectior
Construct a gene set collection of CollectionType entities (e.g., pathways for KEGGCollection, protein families for PfamCollection) implied by the map found in annotation(idType). Use only those identifiers in object. If setType is a CollectionIdType and length(ids(setType))> the gene set collection is filtered to contain only those sets implied by the ids.

signature(object="character", idType="AnnotationIdentifier", setType="PfamCollec
Construct a gene set collection by mapping all values in object to PfamIds found in the PFAM map implied by idType.

signature(object="character", idType="AnnotationIdentifier", setType="PrositeCol
Construct a gene set collection by mapping all values in object to ipi_ids found in the PFAM map implied by idType.

signature(object="character", idType="AnnotationIdentifier", setType="ChrlocColl
Construct a gene set collection by mapping all values in object to chromosome, strand, and position information found in the map implied by idType.

signature(object="ExpressionSet", idType="missing", setType="CollectionType"),si
Construct a gene set collection using the annotation and featureNames of object to identify elements for CollectionType gene sets (e.g., pathways for KEGGCollection, protein families for PfamCollection) implied by object. The gene set collection contains only those AnnotationIdentifiers found in featureNames(object); if setType is a CollectionIdType and length(ids(setType))>0, the gene set collection is further filtered to contain only those sets implied by the ids.

signature(object="ExpressionSet", idType="missing", setType="GOCollection")
Construct a gene set collection using the annotation and featureNames of object to identify GO pathways implied by object. The map between featureNames and GO pathway identifiers is derived from the GO2PROBE table of the annotation package of object. The gene set collection contains only those AnnotationIdentifiers found in featureNames(object). The evidenceCode of GOCollection can be used to restrict the pathways seleted to those with matching evidence codes.

signature(object="ExpressionSet", idType="missing", setType="PfamCollection")
Construct a gene set collection by mapping all values in featureNames(object) to PfamIds found in the PFAM map implied by idType=AnnotationIdentifer(annotation(object))

signature(object="ExpressionSet", idType="missing", setType="PrositeCollection")
Construct a gene set collection by mapping all values in featureNames(object) to ipi_id found in the PFAM map implied by idType=AnnotationIdentifer(annotation(object)).

signature(object="ExpressionSet", idType="missing", setType="ChrlocCollection")
Construct a gene set collection by mapping all values in featureNames(object) to chromosome, strand, and position information found in the CHRLOC map implied by idType=AnnotationIdenti

signature(object="missing", idType="AnnotationIdentifier", setType="GOCollectior

`signature(object="GOAllFrame", idType="missing", setType="GOCollection")`
: Construct a gene set collection containing all GO pathways referenced in the `GOALLFrame` provided. Each gene set only those `Identifiers` found in `GOALLFrame`. The `ontology` of each `GOALLFrame` GO ID will be included in the gene Set of that GO ID .

See Also

`GeneSetCollection`-class

Examples

```
gs1 <- GeneSet(setName="set1", setIdentifier="101")
gs2 <- GeneSet(setName="set2", setIdentifier="102")

## construct from indivdiual elements...
gsc <- GeneSetCollection(gs1, gs2)
## or from a list
gsc <- GeneSetCollection(list(gs1, gs2))

## set names must be unique
try(GeneSetCollection(gs1, gs1))

data(sample.ExpressionSet)
gsc <- GeneSetCollection(sample.ExpressionSet[200:250],
                         setType = GOCollection())

## Not run:
## from KEGG identifiers, for example
library(KEGG.db)
lst <- head(as.list(KEGGEXTID2PATHID))
gsc <- GeneSetCollection(mapply(function(geneIds, keggId) {
    GeneSet(geneIds, geneIdType=EntrezIdentifier(),
            collectionType=KEGGCollection(keggId),
            setName=keggId)
}, lst, names(lst)))

## End(Not run)
```

GeneSet *Methods to construct GeneSet instances*

Description

Use `GeneSet` to construct gene sets from `ExpressionSet`, character vector, or other objects.

Usage

```
GeneSet(type, ..., setIdentifier=.uniqueIdentifier())
```

Arguments

`type`	An argument determining how the gene set will be created, as described in the Methods section.

setIdentifier
: A ScalarCharacter or length-1 character vector uniquely identifying the set.

...
: Additional arguments for gene set construction. Methods have required arguments, as outlined below; additional arguments correspond to slot names GeneSet.

Methods

signature(type = "missing", ..., setIdentifier=.uniqueIdentifier())
: Construct an empty gene set.

signature(type = "character", ..., setIdentifier=.uniqueIdentifier())
: Construct a gene set using identifiers type.

signature(type = "GeneIdentifierType", ..., setIdentifier=.uniqueIdentifier())
: Construct an empty gene set. The gene set has geneIdType created from the GeneIdentifierType of type.

signature(type = "ExpressionSet", ..., setIdentifier=.uniqueIdentifier())
: Construct a gene set from an ExpressionSet. geneIdType is set to AnnotationIdentifier; the annotation field and annotation package of the ExpressionSet are consulted to determine organism, if possible. Short and long descriptions from the ExpressionSet experimentData title and abstract; pub med ids, urls, and contributor are also derived from experimentData.

signature(type = "GOCollection", ..., geneIdType, setIdentifier=.uniqueIdentifie
: Use genes contained in type to create a GeneSet . The required arugment geneIdType must include a package for which an appropriate map (to GO) exists, e.g., EntrezIdentifier('org.Hs.eg.

signature(type = "BroadCollection", ..., urls = character(0), setIdentifier=.uni
: Read XML following the Broad Institute schema and located at urls to create a gene set. The url can be a local file or internet connection, but must contain just a single gene set. See getBroadSets for details.

See Also

GeneSet-class GeneColorSet-class

Examples

```
## Empty gene set
GeneSet()

## Gene set from ExpressionSet
data(sample.ExpressionSet)
gs1 <- GeneSet(sample.ExpressionSet[100:109])

## GeneSet from Broad XML; 'fl' could be a url
fl <- system.file("extdata", "Broad.xml", package="GSEABase")
gs2 <- getBroadSets(fl)[[1]] # actually, a list of two gene sets

## GeneSet from list of gene identifiers
geneIds <- geneIds(gs2) # any character vector would do
gs3 <- GeneSet(geneIds)
## unspecified set type, so...
is(geneIdType(gs3), "NullIdentifier") == TRUE
## update set type to match encoding of identifiers
```

```
geneIdType(gs2)
geneIdType(gs3) <- SymbolIdentifier()
## other ways of accomplishing the same
gs4 <- GeneSet(geneIds, geneIdType=SymbolIdentifier())
gs5 <- GeneSet(SymbolIdentifier(), geneIds=geneIds)
```

`import/export` *Read and write gene sets from Broad or GMT formats*

Description

`getBroadSets` parses one or more XML files for gene sets. The file can reside locally or at a URL. The format followed is that defined by the Broad (below). `toBroadXML` creates Broad XML from `BroadCollection` gene sets.

`toGmt` converts `GeneSetColletion` objects to a character vector representing the gene set collection in GMT format. `getGmt` reads a GMT file or other character vector into a `GeneSetColletion`.

Usage

```
getBroadSets(uri, ...)
toBroadXML(geneSet, con, ...)
asBroadUri(name,
           base="http://www.broad.mit.edu/gsea/msigdb/cards")
getGmt(con, geneIdType=NullIdentifier(),
       collectionType=NullCollection(), sep="\t", ...)
toGmt(x, con, ...)
```

Arguments

`uri`	A file name or URL containing gene sets encoded following the Broad specification. For Broad sets, the uri can point to a MSIGDB.
`geneSet`	A `GeneSet` with `collectionType` `BroadCollection` (to ensure that required information is available).
`x`	A `GeneSetCollection` or other object for which a `toGmt` method is defined.
`con`	A (optional, in the case of `toXxx`) file name or connection to receive output.
`name`	A character vector of Broad gene set names, e.g., `c('chr16q', 'GNF2_TNFSF10')`.
`base`	Base uri for finding Broad gene sets.
`geneIdType`	A constructor for the type of identifier the members of the gene sets represent. See `GeneIdentifierType` for more information.
`collectionType`	A constructor for the type of collection for the gene sets. See `CollectionType` for more information.
`sep`	The character string separating members of each gene set in the GMT file.
`...`	Further arguments passed to the underlying XML parser, particularly `file` used to specify an output `connection` for `toBroadXML`.

Value

`getBroadSets` returns a `GeneSetCollection` of gene sets.

`toBroadXML` returns a character vector of a single `GeneSet` or, if `con` is provided, writes the XML to a file.

`asBroadUri` can be used to create URI names (to be used by `getBroadSets` of Broad files.

`getGmt` returns a `GeneSetCollection` of gene sets.

`toGmt` returns character vectors where each line represents a gene set. If `con` is provided, the result is written to the specified connection.

Note

Actual Broad XML files differ from the DTD (e.g., an implied ',' separator between genes in a set); we parse to and from files as they exists the actual files.

Author(s)

Martin Morgan <mtmrogan@fhcrc.org>

References

http://www.broad.mit.edu/gsea/

See Also

`GeneSetCollection` `GeneSet`

Examples

```
## 'fl' could also be a URI
fl <- system.file("extdata", "Broad.xml", package="GSEABase")
gss <- getBroadSets(fl) # GeneSetCollection of 2 sets
names(gss)
gss[[1]]

## Not run:
## Download 'msigdb_v2.5.xml' or 'c3.all.v2.5.symbols.gmt' from the
## Broad, http://www.broad.mit.edu/gsea/downloads.jsp#msigdb, then
gsc <- getBroadSets("/path/to/msigdb_v.2.5.xml")
types <- sapply(gsc, function(elt) bcCategory(collectionType(elt)))
c3gsc1 <- gsc[types == "c3"]
c3gsc2 <- getGmt("/path/to/c3.all.v2.5.symbols.gmt",
                 collectionType=BroadCollection(category="c3"),
                 geneIdType=SymbolIdentifier())

## End(Not run)

fl <- tempfile()
toBroadXML(gss[[1]], con=fl)
noquote(readLines(fl))
unlink(fl)

## Not run:
toBroadXML(gss[[1]]) # character vector
```

```
## End(Not run)

fl <- tempfile()
toGmt(gss, fl)
getGmt(fl)
unlink(fl)
```

`getOBOCollection` *Read OBO-specified Gene Ontology Collections*

Description

`getOBOCollection` parses a uri (file or internet location) encoded following the OBO specification defined by the Gene Onotology consortium.

Usage

```
getOBOCollection(uri, evidenceCode="ANY", ...)
```

Arguments

`uri`	A file name or URL containing gene sets encoded following the OBO specification.
`evidenceCode`	A character vector of evidence codes.
`...`	Further arguments passed to the `OBOCollection` constructor.

Value

`getOBOCollection` returns an `OBOCollection` of gene sets. The gene set is constructed by parsing the file for `id` tags in `TERM` stanzas. The parser does not currently support all features of OBO, e.g., the ability to import additional files.

Author(s)

Martin Morgan <mtmrogan@fhcrc.org>

References

http://www.geneontology.org

See Also

`OBOCollection`, `OBOCollection`

Examples

```
## 'fl' could also be a URI
fl <- system.file("extdata", "goslim_plant.obo", package="GSEABase")
getOBOCollection(fl) # GeneSetCollection of 2 sets

## Not run:
## Download from the internet
fl <- "http://www.geneontology.org/GO_slims/goslim_plant.obo"
getOBOCollection(fl, evidenceCode="TAS")

## End(Not run)
```

`goSlim-methods` *Methods for Function goSlim in Package 'GSEABase'*

Description

These methods summarize the gene ontology terms implied by the `idSrc` argument into the GO terms implied by the `slimCollection` argument. The summary takes identifiers in `idSrc` and determines all GO terms that apply to the identifiers. This full list of GO terms are then classified for membership in each term in the `slimCollection`.

The resulting object is a data frame containing the terms of `slimCollection` as row labels, counts and frequencies of identifiers calssified to each term, and an abbreviated term description.

An identifier in `idSrc` can expand to serveral GO terms, and the GO terms in `slimCollection` can imply an overlapping hierarchy of terms. Thus the resulting summary can easily contain more counts than there are identifiers in `idSrc`.

Usage

```
goSlim(idSrc, slimCollection, ontology, ..., verbose=FALSE)
```

Arguments

`idSrc`	An argument determining the source of GO terms to be mapped to slim terms. The source might be a `GOCollection` of terms, or another object (e.g., ExpressionSet) for which the method can extract GO terms.
`slimCollection`	An argument containing the GO slim terms.
`ontology`	A character string nameing the ontology to be consulted when identifing slim term hierarchies. One of 'MF' (molecular function), 'BP' (biological process), 'CC' (cellular compartment).
`...`	Additional arguments passed to specific emthods.
`verbose`	Logical influencing whether messages (primarily missing GO terms arising during creation of the slim hierarchy) are reported.

Methods

idSrc="GOCollection", slimCollection="GOCollection", ontology="character", ..., verbose=FALSE
Classify `idSrc` GO terms into `slimCollection` categories. The hierarchy of terms included for each term is from the ontology (MF, BP, or CC) specified by `ontology`. `verbose` informs about, e.g., GO terms that are not found.

idSrc="ExpressionSet", slimCollection="GOCollection", ontology="character", ..., verbose=FALSE
Determine the (unique) GO terms implied by feature names in `idSrc` (using the annotation map identified in `annotation(idSrc)`).

Examples

```
myIds <- c("GO:0016564", "GO:0003677", "GO:0004345", "GO:0008265",
           "GO:0003841", "GO:0030151", "GO:0006355", "GO:0009664",
           "GO:0006412", "GO:0015979", "GO:0006457", "GO:0005618",
           "GO:0005622", "GO:0005840", "GO:0015935", "GO:0000311")
myCollection <- GOCollection(myIds)
fl <- system.file("extdata", "goslim_plant.obo", package="GSEABase")
slim <- getOBOCollection(fl)
goSlim(myCollection, slim, "MF")
data(sample.ExpressionSet)
goSlim(sample.ExpressionSet, slim, "MF", evidenceCode="TAS")
```

GSEABase-package *Gene set enrichment data structures and methods*

Description

This package provides classes and methods to support Gene Set Enrichment Analysis (GSEA). The GeneSet class provides a common data structure for representing gene sets. The GeneColorSet class allows genes in a set to be associated with phenotypes. The GeneSetCollection class facilitates grouping together a list of related gene sets. The GeneIdentifierType class hierarchy reflects how genes are represented (e.g., Entrez versus symbol) in the gene set. mapIdentifiers provides a way to convert identifiers in a set from one type to another. The CollectionType class hierarchy reflects how the gene set was made, and can order genes into distinct sets or collections.

Author(s)

Written by Martin Morgan, Seth Falcon, Robert Gentleman. Maintainer: Biocore Team c/o BioC user list <bioconductor@stat.math.ethz.ch>

See Also

GeneSet, GeneColorSet GeneSetCollection

Examples

```
example(GeneSet)
```

incidence-methods *Methods for Constructing Incidence Matricies Between GeneSets*

Description

An incidence matrix summarizes shared membership of gene identifiers across (pairs of) gene sets.

Methods

The return value is a matrix with rows representing gene sets and columns genes.

All additional arguments `...` are of the same class as `x`. The incidence matrix contains elements 0 (genes not present) or 1 (genes present).

`signature(x="GeneSet", ...)` `signature(x="GeneColorSet", ...)` `signature(x = "GeneSet`

Additional arguments `...` can be of class `GeneSetCollection` or `GeneSet`. The incidence matrix contains elements 0 (genes not present) or 1 (genes present).

Examples

```
fl <- system.file("extdata", "Broad.xml", package="GSEABase")
gss <- getBroadSets(fl) # GeneSetCollection of 2 sets
## From one or more GeneSetCollections...
imat <- incidence(gss)
dim(imat)
imat[,c(1:3,ncol(imat)-3+1:3)]

## .. or GeneSets
imat1 <- incidence(gss[[1]], gss[[2]], gss[[1]])
imat1[,1:5]
```

mapIdentifiers-methods *Methods for Function mapIdentifiers in Package 'GSEABase'*

Description

These methods convert the genes identifiers of a gene set from one type to another, e.g., from `EntrezIdentifier` to `AnnotationIdentifier`. Methods can be called directly by the user; `geneIdType<-` provides similar functionality. `verbose=TRUE` produces warning messages when maps between identifier types are not 1:1, or a map has to be constructed on the fly (this situation does not apply when using the DBI-based annotation packages).

Methods

The following methods are defined on `what="GeneSet"`:

what = "ANY", to = "ANY", from = "ANY", verbose=FALSE This method warns of attempts to map `from` and `to` the same type, or generates an error if no suitable `mapIdentifiers` methods are available.

what = "GeneSet", to = "GeneIdentifierType", from = "missing", verbose=FALSE This method will re-dispatch to a method with signature `signature(what=what, to=to, from=geneIdType(what` and is present so that a user can call `mapIdentifiers` without providing an explicit `from` argument.

what = "GeneSet", to = "GeneIdentifierType", from = "NullIdentifier", verbose=FALSE This maps a gene set from gene identifiers represented by the `NullIdentifier` type (i.e., no type associated with the genes) to gene identifiers represent by any class derived from `GeneIdentifierType`.

what = "GeneSet", to = "NullIdentifier", from = "GeneIdentifierType", verbose=FALSE This maps a gene set from gene identifiers represented by any `GeneIdentifierType` type to one represented by the `NullIdentifier` (i.e., no type associated with the genes).

what = "GeneSet", to = "GeneIdentifierType", from = "environment", verbose=FALSE Maps identifiers found in `what` to the type described by `to`, using the map (key-value pairs) found in `from`.

what = "GeneSet", to = "GeneIdentifierType", from = "AnnDbBimap", verbose=FALSE Maps identifiers found in `what` to the type described by `to`, using the map (key-value pairs) found in `from`.

The following methods are defined for `what=GeneColorSet`. These methods map gene- and phenotype color appropriately, and fail if coloring of gene identifiers involved in several-to-1 mappings conflict.

what = "GeneColorSet", to = "GeneIdentifierType", from = "missing", verbose=FALSE This method will re-dispatch to a method with signature `signature(what=what, to=to, from=geneIdType(what))`, and is present so that a user can call `mapIdentifiers` without providing an explicit `from` argument.

what = "GeneColorSet", to = "GeneIdentifierType", from = "NullIdentifier", verbose=FALSE This maps a gene set from gene identifiers represented by the `NullIdentifier` type (i.e., no type associated with the genes) to gene identifiers represent by any class derived from `GeneIdentifierType`.

what = "GeneColorSet", to = "NullIdentifier", from = "GeneIdentifierType", verbose=FALSE This maps a gene set from gene identifiers represented by any `GeneIdentifierType` type to one represented by the `NullIdentifier` (i.e., no type associated with the genes).

what = "GeneColorSet", to = "GeneIdentifierType", from = "environment", verbose=FALSE This method is not implemented, and exists to stop incorrect application of the `GeneSet` method.

what = "GeneColorSet", to = "GeneIdentifierType", from = "AnnDbBimap", verbose=FALSE This method is not implemented, and exists to stop incorrect application of the `GeneSet` method.

A method exists for `what="GeneSetCollection"`:

what = "GeneSetCollection", to = "GeneIdentifierType", from = "missing", verbose = FALSE Map each gene set in `what` to gene identifier type `to`, using methods described above.

OBOCollection-class *Class "OBOCollection"*

Description

`OBOCollection` extends the `GOCollection` class, and is usually constructed from a file formated following the OBO file format. See `CollectionType` for general use of collections with gene sets.

Objects from the Class

Objects are instantiated with calls to `OBOCollection` or `getOBOCollection`.

Slots

`OBOCollection` extends `GOCollection` and `OBOCollection` has the following additional slots (these slots are NOT meant to be manipulated directly by the user):

`.stanza`: A `data.frame` representing the stanzas present in an OBO file. Row names of the data frame are unique stanza identifiers. The `value` column contains the stanza name (e.g., 'Term', i.e., the stanza name associated with a GO identifier).

`.subset` A `data.frame` representing (optional) subsets defined in the collection. Subsets are defined in the header of an OBO file with a subsetdef tag. Row names of the data frame are the subsetdef names; the `value` column contains the subset definition.

`.kv` A `data.frame` representing key-value pairs in the OBO source file. The row names of the data frame correspond to lines in the OBO file. The `stanza_id` column indexes the row of `.stanza` describing the stanza in which the key-value pair occured. The remaining columns (`key`, `value`) contain the parsed key and value.

Methods

`OBOCollection` has the following methods, in addition to those inherited from `GOCollection`.

These methods list and select subsets of `OBOCollection`:

subsets `signature(object="OBOCollection", display="named")`: return a character vector of subsets present in `object`. Valid values for `display` are 'named' (a named character vector, with names equal to the names of the subsets and values the descriptions), 'full' (a character vector of name and description, with each pair formated into a single entry as "name (description)"), 'key' (subset names), or 'value' (subset descriptions).

[`signature(object="OBOCollection", i="character", j="missing", ...)`: return an `OBOCollection` by selecting just those subsets whose name matches the string(s) in `i`. This method calls the `[,GOCollection` method so, e.g., `evidenceCode` can be used to restricts which evidence codes the collection will identify.

These methods coerce to and from `OBOCollection`:

as `signature(object="OBOCollection", "graphNEL")`: create a directed graph with nodes generated from `ids(object)` and edges from `is_a` relations of `object`.

as `signature(object="graphNEL", "OBOCollection")`: create an `OBOCollection` with `ids` from the graph nodes, and edges from `inNodes(object)`.

Author(s)

Martin Morgan <mtmorgan@fhcrc.org>

References

http://www.geneontology.org for details of the OBO format.

See Also

OBOCollection constructor; CollectionType classes.

Examples

```
fl <- system.file("extdata", "goslim_plant.obo", package="GSEABase")
obo <- getOBOCollection(fl)
obo
subsets(obo)
obo["goslim_plant", evidenceCode="TAS"]
g <- as(obo["goslim_goa"], "graphNEL")
if (interactive() && require("Rgraphviz")) {
    plot(g)
}
```

Index