Package 'InterMineR'

October 14, 2021

Title R Interface with InterMine-Powered Databases

Version 1.14.1

Date 2016-01-05

Author Bing Wang, Julie Sullivan, Rachel Lyne, Konstantinos Kyritsis, Celia Sanchez

Maintainer InterMine Team <r.lyne@gen.cam.ac.uk>

Description Databases based on the InterMine platform such as FlyMine, modMine (modENCODE), RatMine, YeastMine, HumanMine and TargetMine are integrated databases of genomic, expression and protein data for various organisms. Integrating data makes it possible to run sophisticated data mining queries that span domains of biological knowledge. This R package provides interfaces with these databases through webservices. It makes most from the correspondence of the data frame object in R and the table object in databases, while hiding the details of data exchange through XML or JSON.

Depends R (>= 3.4.1)

License LGPL

LazyData true

VignetteBuilder knitr

Imports Biostrings, RCurl, XML, xml2, RJSONIO, sqldf, igraph, httr, S4Vectors, IRanges, GenomicRanges, SummarizedExperiment, methods

Suggests BiocStyle, Gviz, knitr, rmarkdown, GeneAnswers, GO.db, org.Hs.eg.db

BugReports https://github.com/intermine/intermineR/issues

biocViews GeneExpression, SNP, GeneSetEnrichment, DifferentialExpression, GeneRegulation, GenomeAnnotation, GenomeWideAssociation, FunctionalPrediction, AlternativeSplicing, ComparativeGenomics, FunctionalGenomics, Proteomics, SystemsBiology, Microarray, MultipleComparison, Pathways, GO, KEGG, Reactome, Visualization

RoxygenNote 7.1.1

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

git_branch RELEASE_3_13

git_last_commit 1a090a5

git_last_commit_date 2021-05-27

Date/Publication 2021-10-14

R topics documented:

InterMineR-package . . . 3
convertToGeneAnswers . . . 3
convertToGRanges . . . 5
convertToRangedSummarizedExperiment . . . 8
doEnrichment . . . 10
getDatasets . . . 13
getGeneIds . . . 15
getModel . . . 16
getRelease . . . 17
getTemplateQuery . . . 18
getTemplates . . . 19
getVersion . . . 20
getWidgets . . . 20
initInterMine . . . 21
InterMineR-class . . . 22
InterMineR-methods . . . 23
listDatasets . . . 24
ListManager-class . . . 25
ListManager-methods . . . 26
listMines . . . 28
list_manager . . . 29
newQuery . . . 30
PL_DiabetesGenes . . . 31
PL_FlyTF_site_specific_TFs . . . 32
runQuery . . . 32
Service-class . . . 35
setConstraints . . . 36
setQuery . . . 38
simplifyResult . . . 40
summary . . . 42

Index **44**

InterMineR-package *R Interface with InterMine-powered databases*

Description

InterMine-powered databases such as FlyMine, modENCODE, RatMine, YeastMine, HumanMine and TargetMine are integrated databases of genomic, expression and protein data for various organisms. Integrating data makes it possible to run sophisticated data mining queries that span domains of biological knowledge. This R package provides interface with these databases through webservices. It makes most from the correspondence of the data frame object in R and the table object in databases while hiding the details of data exchange through XML or JSON.

Details

Package:	InterMineR
Type:	Package
Version:	0.99.4
Date:	2016-01-05
License:	LGPL
Depends:	Biostrings, RCurl, XML, RJSONIO, sqldf, igraph
Suggests:	Gviz

Author(s)

InterMine Team

Maintainer: InterMine Team <info@intermine.org>

References

http://intermine.readthedocs.org/en/latest/web-services/

convertToGeneAnswers *Convert InterMineR enrichment analysis results to GeneAnswers object*

Description

convertToGeneAnswers constitutes a wrapper function for converting the results of doEnrichment function to a GeneAnswers-class object. This way the user can utilize the functions of GeneAnswers package to visualize the results of InterMineR enrichment analysis.

Usage

```
convertToGeneAnswers(
  enrichmentResult,
  geneInput,
  geneInputType,
  geneExprProfile,
  annLib,
  categoryType,
  enrichCategoryChildName
)
```

Arguments

enrichmentResult	a list containing the results of doEnrichment function.
geneInput	a data.frame containing the gene identifiers used for the InterMineR enrichment analysis, and possible values associated with them.
geneInputType	a character string specifying the InterMineR gene identifier type of the values assigned to geneInput argument. InterMineR identifiers for each Mine can be retrieved with getModel function.
geneExprProfile	a data.frame containing gene expression information (optional)
annLib	name of given annotation library file or user provided annotation list.
categoryType	name of given annotation category or NULL for user provided annotation list.
enrichCategoryChildName	a character string specifying the InterMIneR annotation category identifier. This argument must be assigned manually if: 1) 'enrichIdentifier' column is missing from the widgets of the Mine used by doEnrichment, 2) the value of 'enrichIdentifier' for the enrichment widget used by doEnrichment is NA. InterMineR identifiers for each Mine can be retrieved with getModel function.

Value

a GeneAnswers-class object.

Author(s)

InterMine Team

References

https://bioconductor.org/packages/release/bioc/html/GeneAnswers.html

See Also

doEnrichment, GeneAnswers-class, getWidgets, getModel

Examples

```
#The example code is hashed out as it requires an API key to run
# load human genes which are associated with Diabetes (retrieved from HumanMine)
#data("PL_DiabetesGenes")

# get Gene.primaryIdentifiers (ENTREZ Gene identifier)
#hsa_gene_entrez = as.character(PL_DiabetesGenes$Gene.primaryIdentifier)

# perform enrichment analysis with InterMineR
#im.human = initInterMine(listMines()["HumanMine"], 'Your token') #change here to your token
#class(im.human)
#hsa_enrichResult = doEnrichment(
#  im.human,
#  genelist = "PL_DiabetesGenes",
#  widget = "go_enrichment_for_gene",
#  correction = "Benjamini Hochberg"
#)

# convert InterMineR enrichment analysis results to a GeneAnswers object
# Do not run unless you have installed and loaded GeneAnswers package!

# load GeneAnswers
#library(GeneAnswers)

#hsa_geneanswers = convertToGeneAnswers(
#  enrichmentResult = hsa_enrichResult,
#  geneInput = data.frame(GeneID = as.character(hsa_gene_entrez),
#                         stringsAsFactors = FALSE),
#  geneInputType = "Gene.primaryIdentifier",
#  annLib = 'org.Hs.eg.db',
#  categoryType = "GO"
  # enrichCategoryChildName = "Gene.goAnnotation.ontologyTerm.parents.identifier"
#)
```

---

convertToGRanges *Convert InterMineR retrieved genomic information to objects of class GRanges*

Description

convertToGRanges constitutes a wrapper function for converting genomic locations and their respective annotations to objects of the GRanges class.

Usage

```
convertToGRanges(
  dataset,
  seqnames,
```

```
  start,
  end,
  names,
  strand,
  columnsAsMetadata = NULL,
  listAsMetadata = NULL,
  seqnamesInterMineChromosome = TRUE
)
```

Arguments

dataset	a data.frame containing all the genomic information retrieved by InterMineR. Its columns are used to create the GRanges object.
seqnames	the name of a column from dataset or a character vector of length equal to the number of rows of the dataset. Defines the values assigned as seqnames to the GRanges object.
start	the name of a column from dataset or a vector of length equal to the number of rows of the dataset. Defines the genomic coordinates assigned as start to the ranges argument of the GRanges object.
end	the name of a column from dataset or a vector of length equal to the number of rows of the dataset. Defines the genomic coordinates assigned as end to the ranges argument of the GRanges object.
names	the name of a column from dataset or a character vector of length equal to the number of rows of the dataset. Defines the values assigned as names to the ranges argument of the GRanges object.
strand	the name of a column from dataset or a character vector of length equal to the number of rows of the dataset. Defines the values assigned as strand information to the GRanges object.
columnsAsMetadata	a character vector containing the names of the dataset columns that are passed as metadata in the GRanges object.
listAsMetadata	a list of vectors, each of which has length equal to the number of rows of the dataset. The values of the list are passed as metadata to the GRanges object.
seqnamesInterMineChromosome	a logical value indicating whether the values passed as seqnames are InterMineR chromosome.primaryIdentifiers (e.g. "2R", "3R", "X") or not.

Details

The InterMineR package provides a flexible interface to the InterMine web services, which allow for rapid retrieval of various genomic information.

convertToGRanges function is designed to facilitate the conversion of genomic locations and their respective annotations, in the format by which they are retrieved using InterMineR, to an object of the GRanges class.

Value

An object of the GRanges class containing genomic locations and annotations retrieved by InterMineR queries.

Author(s)

InterMine Team

See Also

GRanges, convertToRangedSummarizedExperiment

Examples

```
# get FlyMine
im.fly = initInterMine(listMines()["FlyMine"])

# modify template query for Transcription Factor (TF) Binding sites
qTF_Binding = getTemplateQuery(im.fly,"ChromLocation_TFBindingSiteLocationGeneFactor")

qTF_Binding$where[[4]]$value = "1000000"
qTF_Binding$where[[5]]$value = "20000000"

rTF_Binding = runQuery(im.fly, qTF_Binding)

# assign random values for strand of the genomic location retrieved, in InterMine format
rTF_Binding$gene.strand = sample(c("1", "-1", ""), nrow(rTF_Binding), replace = TRUE)

# convert to GRanges object
test = convertToGRanges(
  dataset = rTF_Binding,
  seqnames = rTF_Binding$TFBindingSite.chromosome.primaryIdentifier,
  names = rTF_Binding$TFBindingSite.factor.name,
  start = rTF_Binding$TFBindingSite.chromosomeLocation.start,
  end = rTF_Binding$TFBindingSite.chromosomeLocation.end,
  strand = "gene.strand",
  columnsAsMetadata = c(
    "TFBindingSite.gene.regulatoryRegions.dataSets.dataSource.name",
    "TFBindingSite.factor.primaryIdentifier"),
  listAsMetadata = list(
    c(factor.primaryIdentifier = rTF_Binding$TFBindingSite.factor.primaryIdentifier)
  )
)

# check results
test
```

convertToRangedSummarizedExperiment
Convert experimental results retrieved by InterMineR queries to an object of the RangedSummarizedExperiment *class*

Description

convertToRangedSummarizedExperiment constitutes a wrapper function for converting genomic information and experimental data from InterMine to an object of the RangedSummarizedExperiment class.

Usage

```
convertToRangedSummarizedExperiment(
  im,
  dataset,
  SampleColumn,
  GeneColumn,
  ValueColumn,
  OrganismValue,
  colsForSampleMetadata,
  exonsForRowRanges = FALSE
)
```

Arguments

im	a Service object containing the base URL and API token, created by initInterMine.
dataset	a data.frame retrieved with InterMineR queries that contains experimental data from high-throughput assays.
SampleColumn	a character string or an integer indicating which column of the dataset contains the samples.
GeneColumn	a character string or an integer indicating which column of the dataset contains the genes.
ValueColumn	a character string or an integer indicating which column of the dataset contains the experimental data.
OrganismValue	a character string with the name of the organism from which the genomic information are retrieved.
colsForSampleMetadata	an integer vector indicating the columns of the dataset which will be assigned as sample metadata in the colData argument of the RangedSummarizedExperiment.
exonsForRowRanges	a logical value indicating whether the rowRanges argument of the RangedSummarizedExperiment should be assigned with a GRangeslist containing only annotations about the genes (default) or annotations about all exons of each gene.

Details

The InterMineR package provides a flexible interface to InterMine web services, which allow for rapid retrieval of various genomic information.

convertToRangedSummarizedExperiment function facilitates the conversion of genomic information and experimental data from high-throughput assays, which are retrieved by using InterMineR queries, to an object of the RangedSummarizedExperiment class.

It is noteworthy that the reshape function is used to convert the experimental data from the InterMineR format (long format) to the matrix (wide format) assigned to the assays argument of the SummarizedExperiment.

Value

an object of the RangedSummarizedExperiment class containing genomic information and experimental data of high-throughput assays, which are retrieved with the InterMineR queries system.

Author(s)

InterMine Team

References

SummarizedExperiment for Coordinating Experimental Assays, Samples, and Regions of Interest

See Also

RangedSummarizedExperiment, link{convertToGRanges}, link{initInterMine}

Examples

```
# 10 Drosophila melanogaster genes of interest
Drosophila.genes = c("BEAF-32", "Antp", "bcd", "caup", "tup", "E2f2", "dsx", "so", "toy", "Lim1")

# retrieve microarray time course experimental data for Drosophila.genes
# get FlyMine instance
im.fly = initInterMine(listMines()["FlyMine"])

# get FlyMine microarray time course template query
queryForData = getTemplateQuery(im.fly, "Gene_TimeCourseExpression")

test.data = list(NULL)
ind.null = c()

for(i in seq(length(Drosophila.genes))){
  # set value in gene constraint
  queryForData$where[[3]]$value = as.character(Drosophila.genes[i])

 # run query and save the results of genes that exist in the Microarray time course dataset
  r = runQuery(im.fly, queryForData)

  ind.null = c(ind.null, is.null(r))
```

```
  test.data[[i]] = r

}

# remove genes for which no experimental data were retrieved
test.data = test.data[which(!ind.null)]

# rbind data together
test.data = do.call(rbind, test.data)

# using integer index for columns in arguments
test1 = convertToRangedSummarizedExperiment(
  im = im.fly,
  dataset = test.data,
  SampleColumn = 2,
  GeneColumn = 1,
  ValueColumn = 3,
  OrganismValue = "Drosophila melanogaster",
  colsForSampleMetadata = 4:7,
  exonsForRowRanges = TRUE
)

test1

# using directly column names in arguments
test2 = convertToRangedSummarizedExperiment(
  im = im.fly,
  dataset = test.data,
  SampleColumn = "Gene.microArrayResults.assays.sample2",
  GeneColumn = "Gene.symbol",
  ValueColumn = "Gene.microArrayResults.value",
  OrganismValue = "Drosophila melanogaster",
  colsForSampleMetadata = 4:7,
  exonsForRowRanges = TRUE
)

test2
```

`doEnrichment` *Perform enrichment analysis*

Description

Retrieve enrichment analysis results from InterMine platform. Enrichment widgets provide a statistical summary of what makes a list distinct from the background population over a certain domain. They return a list of members of the domain ranked by p-value (low to high).

Usage

```
doEnrichment(
  Service-class,
  genelist = NULL,
  ids = NULL,
  widget = NULL,
  population = NULL,
  maxp = 0.05,
  correction = "Benjamini Hochberg",
  filter = NULL,
  organism = NULL
)
```

Arguments

Service-class	a Service-class object containing the base URL and API token.
genelist	The name of the list to investigate, optional unless ids (identifiers) is NULL.
ids	a character vector containing the identifiers (list of genes, proteins, SNPs, etc.) for the enrichment analysis, optional unless genelist is NULL. Alternatively, a comma-separated character string of InterMine object IDs can be assigned directly to this argument. Use getGeneIds function to retrieve unique Gene.id values for a list of gene identifiers.
widget	The name of the enrichment widget to display. Use getWidgets function to retrieve available enrichment type widgets for the respective Mine.
population	The name of the list to use as the background population.
maxp	The maximum p-value of results to display. The range is 0.0 - 1.0
correction	The error correction algorithm to use. Possible options are "Benjamini Hochberg" (default), "Holm-Bonferroni", "Bonferroni" or "None".
filter	An optional filter that some widgets accept. Use getWidgets function to retrieve available filters of the respective enrichment widget.
organism	a character string defining the name of the organism (e.g. "Homo sapiens"). This argument is optional and can be used to limit the analysis to the list of identifiers for a specific organism. Can be useful for InterMine instances that contain data from multiple organisms.

Details

The public gene lists for each Mine are available at the websites obtained by the following command: paste0(listMines()["Mine_Name"], "/bag.do?subtab=view")

Each type of enrichment widget can be applied to a gene list with specific annotation. To apply the enrichment analysis to the appropriate list of genes, use the targets column of the respective enrichment widgets as they are retrieved by getWidgets function.

Value

doEnrichment function returns a list containing the following values:

`data`	A data.frame containing the results of the enrichment analysis performed in InterMine platform. The statistically significant results are ordered by increasing p-values
`populationCount`	a numeric value indicating the size of the reference population
`notAnalyzed`	a numeric value indicatng the number of input features that were not included in the enrichment analysis
`im`	the list containing the base URL and API token
`parameters`	a character vector containing all the parameters used for the enrichment analysis, except im

Author(s)

InterMine Team

See Also

`getWidgets`, `getGeneIds`

Examples

```
# FlyMine
# The example has been hashed out as it requires an API token to run
# im.fly = initInterMine(listMines()["HumanMine"],"Your token") #change here to your token
# class(im.fly)
# enrichResults.FlyMine <- doEnrichment(
# im.fly,
# genelist = "PL FlyAtlas_brain_top",
#  #genelist = 'PL FlyTF_site_specific_TFs',
#  widget = "go_enrichment_for_gene"
#)

# use ids instead of genelist
#data("PL_FlyTF_site_specific_TFs")
#enrichResults.FlyMine <- doEnrichment(
#  im.fly,
  #genelist = 'PL FlyTF_site_specific_TFs',
#  ids = PL_FlyTF_site_specific_TFs$Gene.primaryIdentifier,
#  widget = "go_enrichment_for_gene"
#)

# HumanMine
#im.human = initInterMine(listMines()["HumanMine"],"Your token") #change here to your token
#class(im.human)
#introduce the name of a genelist that you have created or not
#enrichResults.HumanMine <- doEnrichment(
#  im.human,
```

```
#  genelist = "copy_of_intersected",
#  widget = "go_enrichment_for_gene"
#  )

# use ids instead of genelist
#data("PL_DiabetesGenes")
#enrichResults.HumanMine <- doEnrichment(
#  im.human,
  #genelist = "PL_DiabetesGenes",
#  ids = PL_DiabetesGenes$Gene.primaryIdentifier,
#  widget = "go_enrichment_for_gene"
#)
```

`getDatasets` *Retrieve information about the available datasets of a Mine instance.*

Description

This function retrieves information about the available datasets of a mine instance. It is necessary to specify the type of feature (e.g. Gene, TFBindingSite) for this mine, and a value of this feature OR of one of its child_name values.

Usage

```
getDatasets(
  im,
  type,
  child_name,
  value,
  op
)
```

Arguments

`im`	a Service object containing the base URL and API token, created by `initInterMine`.
`type`	a character string defining the type of feature for which the information about the available datasets will be retrieved. Use the `listDatasets` function to retrieve types with available datasets on each mine.
`child_name`	a character string defining the child_name values of type, which will be used to retrieve the information about the available datasets. Use `getModel` function to retrieve the available child_name values for each type.
`value`	a character string defining the value of the child_name.
`op`	a character string defining the op argument.

Details

The getDatasets function uses an InterMineR query to retrieve the information of the available datasets for:

1. a specific mine (e.g. FlyMine)

2. a specific type of feature (e.g. Gene, TFBindingSite) for this mine, and

3. a specific value of this feature OR of one of its child_name values.

For all genes of Drosophila melanogaster (Gene.organism = Drosophila melanogaster) OR for a specific gene (Gene LOOKUP PPARG)

Value

a data.frame containing information about about the available datasets for a specific type of feature (e.g. Gene) on a specific mine instance (HumanMine, FlyMine, etc.).

Author(s)

InterMine Team

See Also

listDatasets, getModel

Examples

```
# Define mines
im.fly = initInterMine(listMines()["FlyMine"])
im.human = initInterMine(listMines()["HumanMine"])

# get information about all available datasets of Drosophila melanogaster genes
dme_gene_datasets = getDatasets(
  im = im.fly,
  type = "Gene",
  child_name = "organism.name",
  value = "Drosophila melanogaster",
  op = "="
)

# get available datasets for PPARG gene from FlyMine
pparg_related = getDatasets(
  im = im.fly,
  type = "Gene",
  value = "PPARG",
  op = "LOOKUP"
)

# get available datasets for PPARG gene from HumanMine
pparg_related.2 = getDatasets(
  im = im.human,
  type = "Gene",
  value = "PPARG",
```

```
  op = "LOOKUP"
)
```

`getGeneIds` *Get Gene.id values for a list of gene identifiers*

Description

This function takes as input a list of gene identifiers and retrieves their unique Gene.id values for a specific organism.

Usage

```
getGeneIds(im, genes, organism)
```

Arguments

`im`	a Service object containing the base URL and API token, created by `initInterMine`.
`genes`	a character vector containing the gene identifiers.
`organism`	a character string for the name of the Organism.

Details

This function is designed to work in conjuction with the `doEnrichment` function.

Value

getGeneIds returns a list containing a data.frame of unique gene identifiers as well as the Gene.id values as a comma-separated string, ready to be used as input for the ids argument of the doEnrichment function.

Furthermore, in case of genes that return multiple identifiers or no identifiers, these are also reported separately in a data.frame and a character vector respectively.

`unique.results`
: data.frame with unique gene identifiers

`doEnrichment.string`
: comma-separated character string of Gene.id values

`multiple.results`
: data.frame with genes returning multiple identifiers

`genes.with.no.results`
: character vector with genes returning no identifiers

Note

doEnrichment.string can be passed as input in the `doEnrichment` function.

Author(s)

InterMine Team

See Also

getWidgets, doEnrichment

Examples

```
# HumanMine and Homo sapiens genes
hsa.genes <- c("ABCC8", "ACE", "AKT2", "APPL1", "AQP2", "AVP", "AVPR2", "BLK", "CAPN10", "CCR5")

list.hsa <- getGeneIds(
  im = initInterMine(listMines()["HumanMine"]),
  organism = "Homo sapiens",
  genes = hsa.genes # must be a character vector!!!
)

# FlyMine and Drosophila melanogaster genes
fly.genes <- c("fkh", "vvl", "BEAF-32", "Antp", "Scr", "abd-A", "bcd", "Ubx", "zen", "ara")

list.fly <- getGeneIds(
  im = initInterMine(listMines()["FlyMine"]),
  organism = "Drosophila melanogaster",
  genes = fly.genes # must be a character vector!!!
)
```

---

getModel *Get the model of InterMine*

Description

Returns a representation of the data model for the mine. This describes the kind of data held, and the properties that data can have. This information can be used to build queries against that data, and to interpret the information received.

Usage

```
getModel(im, timeout = 3)
```

Arguments

im	a Service object containing the base URL and API token, created by initInterMine.
timeout	an integer, representing the number of seconds to wait for the webservice to respond.

Details

The details of the data model for the various mines are available at the websites obtained by running the following command: paste(listMines()$URL, "/tree.do", sep="")

Value

a multi-level list, representating the data model for the mine. The first-level is a list of the InterMine objects (e.g., Gene, Exon). Each second-level list, corresponding to an InterMine object, contains three data.frame objects: attributes, references and collections. Each attribute is a property of the InterMine object. Each reference or collection is itself an InterMine object, acting as a member object of the InterMine object.

Author(s)

InterMine Team

Examples

```
# Retrieve data model for FlyMine  # temporarily removed
# im.fly = initInterMine(listMines()["FlyMine"])

# model = getModel(im.fly)
```

---

getRelease *Get the current release information of InterMine*

Description

Returns a string describing the release of the mine.

Usage

```
getRelease(im, timeout = 3)
```

Arguments

im	a Service object containing the base URL and API token, created by initInterMine.
timeout	an integer, representing the number of seconds to wait for the webservice to respond.

Value

a string, describing the release of the mine.

Author(s)

InterMine Team

Examples

```
im <- initInterMine(listMines()["HumanMine"])

getRelease(im)
```

`getTemplateQuery` *Get the query contained in a template*

Description

Get a template query for a mine. A template contain a saved query with a view and constraint. The user can modify this query to obtain the desired result. The view is a vector containing the output columns of the query. The constraint is a matrix containing the following columns: path (the path of the constraint), op (the constraint operator, one of '=', '!=', 'LOOKUP', 'ONE OF', 'NONE OF', '>', '<', '>=', '<=', 'LIKE'), value (the constraint value), code (the name of the constraint), extraValue (optional, required for LOOKUP constraints).

Usage

```
getTemplateQuery(im, name, timeout=3)
```

Arguments

`im`	a Service object containing the base URL and API token, created by `initInterMine`.
`name`	a string, representing the name of the pre-defined template.
`timeout`	an integer, representing the number of seconds to wait for the webservice to respond.

Value

a list, reprenting the query contained in the pre-defined template. The list should contain at least two elements, view and constrain.

Author(s)

InterMine Team

Examples

```
# Get template queries from HumanMine
im <- initInterMine(listMines()["HumanMine"])

queryGeneIden <- getTemplateQuery(im, "Gene_Identifiers")
```

`getTemplates` *Get the information (name and title) of the templates pre-defined in InterMine*

Description

Get the information (name and title) of the templates pre-defined in InterMine. A template contain a query with fixed set of output columns, and at least one editable constraint, and possibly more.

Usage

```
getTemplates(im, format = "data.frame", timeout = 3)
```

Arguments

`im`	a Service object containing the base URL and API token, created by `initInterMine`.
`format`	a string with values being either "data.frame" or "list", representing the output format of the template information.
`timeout`	an integer, representing the number of seconds to wait for the webservice to respond.

Value

a data.frame or list object, representing the information (name and title) for the pre-define templates in the mine.

Author(s)

InterMine Team

Examples

```
# Get HumanMine template queries
im <- initInterMine(listMines()["HumanMine"])

templates <- getTemplates(im)
```

getVersion *Get the version information of InterMine*

Description

Returns an integer representing the capabilities of the webservice.

Usage

```
getVersion(im, timeout = 3)
```

Arguments

im	a Service object containing the base URL and API token, created by initInterMine.
timeout	an integer, representing the number of seconds to wait for the webservice to respond.

Value

an integer, representing the capabilities of the webservice.

Author(s)

InterMine Team

Examples

```
im <- initInterMine(listMines()["HumanMine"])

getVersion(im)
```

getWidgets *Get the widgets of InterMine*

Description

Returns a representation of the available widgets for the mine. Each entry for the widget includes details of its type, the kind of data it processes, and any filters it accepts.

Usage

```
getWidgets(im)
```

Arguments

im	a Service object containing the base URL and API token, created by initInterMine.

Value

A data.frame containing information about the widgets that are available for each mine.

Note

The names of the widgets of the enrichment type can be passed as arguments to doEnrichment function.

Author(s)

InterMine Team

See Also

getGeneIds, doEnrichment

Examples

```
# Get available FlyMine widgets
FlyMine.widgets <- getWidgets(im = initInterMine(mine = listMines()["FlyMine"]))
# Get available HumanMine widgets
HumanMine.widgets <- getWidgets(im = initInterMine(mine = listMines()["HumanMine"]))
```

---

initInterMine *Initialize the list containing the base URL and API token.*

Description

Initialize the InterMine list with the base URL of the webservice of the database and the API token.

Some resources such as lists are normally privately associated with the individual user that created them and require authentication for access. To access these private resources, each request needs to be authenticated, using an API key token. You can get an API token from the web-app of the service you intend to access: visit the MyMine tab after logging-in and click on API Key.

Usage

```
initInterMine(mine = listMines()["HumanMine"], token="")
```

Arguments

mine	a string, representing the base URL of the webservice of the database.
token	a string, representing the API token in order to use private functions such as list and enrichment.

Value

A Service object containing the base URL and API token.

Author(s)

InterMine Team

References

/urlhttp://intermine.readthedocs.io/en/latest/web-services/

Examples

```
#hashed out example as requires API key
#im <- initInterMine(mine = listMines()["HumanMine"], "TOKEN") #replace TOKEN with your token
#class(im)
```

`InterMineR-class` *InterMineR class contains the input values for performing queries in an InterMine instance.*

Description

InterMineR constitutes a class used to store the information which are required for performing a query for biological data in an InterMine instance. Specifically, it contains information about:

1) the type of data which are to be returned from the InterMine instance,

2) the type of sorting performed on these data, and

3) the constraints used to perform the query for the data of interest.

Creating Objects

Objects can be created using the function `setQuery`.

Slots

name Assign with a character string giving a name to the query. Pre-fixed with "".

description Assign with a character string describing the purpose and the type of data retrieved by the query. Pre-fixed with "".

select a character vector defining the type of data to be returned.

orderBy a list the name of the column and the type of sorting which will be used to order the retrieved data.frame.

where a list containing the constraints used to restrict the query. Each constraint constitutes a list as well.

Details

InterMineR class specifies an object in which the input values of a query can be stored. A single constraint within the object can be assigned with multiple values.

Author(s)

InterMine Team

See Also

setConstraints, setQuery, InterMineR-methods

InterMineR-methods *Methods for accessing* InterMineR-class *objects.*

Description

InterMineR constitutes a class used to store the information which are required for performing a query for biological data in an InterMine instance. The methods for accessing the slots of an InterMineR-class object are presented here.

Methods

Class-specific methods:

`getName(InterMineR-class)`: Access the name slot of `InterMineR-class` object.

`getDescription(InterMineR-class)`: Access the description slot of `InterMineR-class` object.

`getSelect(InterMineR-class)`: Access the select slot of `InterMineR-class` object.

`getOrderBy(InterMineR-class)`: Access the orderBy slot of `InterMineR-class` object.

`getWhere(InterMineR-class)`: Access the where slot of `InterMineR-class` object.

Author(s)

InterMine Team

See Also

setConstraints, setQuery, InterMineR-class

Examples

```
# get mine instance
im.fly = initInterMine(listMines()["FlyMine"])

# get GO_Gene template query
qGO_Gene = getTemplateQuery(im.fly, "GO_Gene")

# create 'InterMineR' object
go.query = setQuery(
  inheritQuery = qGO_Gene
)
```

```
class(go.query)

# access name:
getName(go.query)

# access description:
getDescription(go.query)

# access select:
getSelect(go.query)

# access orderBy:
getOrderBy(go.query)

# access where:
getWhere(go.query)
```

`listDatasets` *Retrieve all types of InterMine features that possess dataSets as a child_name.*

Description

This function retrieves all types of features from the data model of each Mine, that possess dataSets as a child_name. For these types of features, there are available datasets in the respective mine instance.

Usage

```
listDatasets(im)
```

Arguments

`im`	a Service object containing the base URL and API token, created by `initInterMine`.

Value

a character vector containing the all types of features from the data model of the mine, that possess dataSets as a child_name.

Author(s)

InterMine Team

See Also

`getDatasets`

Examples

```
# HumanMine
#listDatasets(im = initInterMine(listMines()["HumanMine"]))

# FlyMine
#listDatasets(im = initInterMine(listMines()["FlyMine"]))
```

`ListManager-class` *ListManager class provides methods to manage list contents and operations.*

Description

ListManager constitutes a class used to store the information required for managing lists contents and performing operations. Specifically, it contains information about:

1) the default list name and description,

2) the different URL endpoints, and

3) the information of the WebService.

Creating Objects

Objects can be created using the function `list_manager`, which is a webservice method.

Slots

name Assign with a character string giving a name to the query. Pre-fixed with "".

DEFAULT_LIST_NAME Assign with a character string, it is used when the names is not specified or the list exists.

DEFAULT_DESCRIPTION a character string that indicates that the list is created with the R client library.

LIST_PATH URL endpoint for storing lists.

INTERSECTION_PATH URL endpoint for intersecting lists.

UNION_PATH URL endpoint for the union of lists.

DIFFERENCE_PATH URL endpoint for the difference of lists.

SUBTRACTION_PATH URL endpoint for the subtraction lists.

mine URL of the an InterMine Webservice.

token API access key.

Details

ListManager class specifies an object in which the the common inputs to make an API request are stored.

Author(s)

InterMine Team

See Also

list_manager, ListManager-methods, webservice-methods

ListManager-methods *Methods for managing and doing operations with* ListManager-class *objects.*

Description

ListManager constitutes a class used to store the information required for managing lists contents and performing operations. The methods that enable to treat with lists and do operations are described below:

Methods

Class-specific methods:

`get_list(ListManager-class, list_name)`: Return a list from the service by name, if it exists.
\itemlist_name the name of an existing list

`delete_lists(ListManager-class, lists)`: Deletes the lists passed as the second argument. Only deletes lists that belong to the user. The second argument need to be of the form c("list_name_1","list_name_2","list_name_3").
\itemlists a list containing the names of the lists to be deleted.

`create_list(ListManager-class, content, list_type, name = NULL, description = NULL, organism = NULL)`: Creates a new list by uploading a set of identifiers.
\itemcontent the result of a query obtained with runQuery or a lists of identifiers.
\itemlist_type the type of objects to include in the list.
\itemname the name for the new list. If none is provided one will be generated.
\itemdescription a description for the list (free text).
\itemorganism the name of the organism.

`intersect(ListManager-class, lists, name = NULL, description = NULL, tags = list())`: Creates new lists which contain only those items which are members of all the source lists.
\itemlists the source lists to be intersected.
\itemname the name for the new list. If none is provided one will be generated.
\itemdescription a description for the list (free text).
\itemtags a set of strings to use as tags.

`union(ListManager-class, lists, name = NULL, description = NULL, tags = list())`: Creates new lists which contain all the members contained in the set of input lists.
\itemlists the source lists to be unified.
\itemname the name for the new list. If none is provided one will be generated.
\itemdescription a description for the list (free text).
\itemtags a set of strings to use as tags.

`difference(ListManager-class, lists, name = NULL, description = NULL, tags = list())`:
Creates new lists which only contain members which are not shared by an even number of lists.

\itemlists the source lists.

\itemname the name for the new list. If none is provided one will be generated.

\itemdescription a description for the list (free text).

\itemtags a set of strings to use as tags.

`subtract(ListManager-class, lefts, rights, name = NULL, description = NULL, tags = list())`:
Creates new lists which contain only those elements which are present in one set of lists, and none of those elements which are present in another set of lists. This is what is typically thought of as subtraction, or more technically, the asymmetric difference of two sets.

\itemlefts the source lists.

\itemrights the substraction lists.

\itemname the name for the new list. If none is provided one will be generated.

\itemdescription a description for the list (free text).

\itemtags a set of strings to use as tags.

Author(s)

InterMine Team

See Also

ListManager-class, webservice-methods

Examples

```
#the example code has been hashed out as an API token is required.
# get ListManager class
#im <- initInterMine(listMines()["HumanMine"],"YOUR TOKEN")
#im.human.list <- list_manager(im)
#class(im.human.list)

# create "diabetesGenes" list, and another one without specifying the name
#query1Diabetes <- setQuery(
#  select = c("Gene.primaryIdentifier", "Gene.symbol"),
#  where = setConstraints(
#    paths = c("Gene.organism.name", "Gene.diseases.name"),
#    operators = c("=", "CONTAINS"),
#    values = list("Homo sapiens","diabetes")
#  )
#)
#query1DiabetesResults <- runQuery(im, query1Diabetes)
#create_list(im.human.list,content = query1DiabetesResults, list_type = "Gene", name = "diabetesGenes")

#query2UpInPancreasConstraint = setConstraints(
#  paths = c("Gene",
#            "Gene.proteinAtlasExpression.level",
#            "Gene.proteinAtlasExpression.level",
#            "Gene.proteinAtlasExpression.tissue.name"),
```

```
#  operators = c("IN", rep("=", 3)),
#  values = list("PL_Pax6_Targets", "Medium", "High", "Pancreas")
#)
#query2UpInPancreas = newQuery(
#  view = c("Gene.primaryIdentifier",
#           "Gene.symbol",
#           "Gene.proteinAtlasExpression.cellType",
#           "Gene.proteinAtlasExpression.level",
#           "Gene.proteinAtlasExpression.tissue.name"
#  ),
#  constraintLogic = "A and (B or C) and D"
#)
#query2UpInPancreas$where <- query2UpInPancreasConstraint
#query2UpInPancreasResults <-  runQuery(im = im, qry = query2UpInPancreas)
#ids_query2UpInPancreasResults <- list(query2UpInPancreasResults["Gene.primaryIdentifier"])
#create_list(im.human.list,ids_query2UpInPancreasResults,list_type="Gene")
#this will create a list with the default name "my_list"

# example of intersect
#intersect(im.human.list, c("diabetesGenes","my_list"),"intersect_list")

#example of union
#union(im.human.list, c("diabetesGenes","my_list"), "union_list")

#example of difference
#difference(im.human.list,c("diabetesGenes","my_list"),"diff_list")

#example of subtract
#subtract(im.human.list,lefts = c("diabetesGenes"), rights = c("my_list"),"subtr_list")

# get "my_list" list
#resp <- get_list(im.human.list, "my_list")

#delete "my_list" and "diabetesGenes" lists
#delete_lists(im.human.list, c("my_list", "diabetesGenes"))
#trying to delete a list not created by the user gives an error
#delete_lists(im.human.list, c("PL_DiabesityGWAS_pval-4"))
#
```

`listMines` *List the available InterMine-powered databases*

Description

InterMine-powered databases such as FlyMine, modENCODE, RatMine, YeastMine, HumanMine and TargetMine are integrated databases of genomic, expression and protein data for various organisms. The function listMines() lists the current available databases.

Usage

```
listMines()
```

Value

A character vector containing the name and the base URL of the web service of the database.

Author(s)

InterMine Team

References

http://registry.intermine.org/

Examples

```
listMines()
```

`list_manager` *A Class for Managing List Content and Operations*

Description

This class may be called itself, and all the useful methods it has are available through the list objects themselves.

Usage

```
list_manager(Service-class)
```

Arguments

`Service-class`	a `Service-class` object containing the base URL and API token.

Details

The public gene lists for each Mine are available at the websites obtained by the following command: paste0(listMines()["Mine_Name"], "/bag.do?subtab=view")

Get a new ListManager to use with this service. This class provides methods to manage list contents and operations.

Value

list_manager function returns an object containing the following values:

`DEFAULT_LIST_NAME`	"my_list"
`DEFAULT_DESCRIPTION`	"List created with R client library"
`URL endpoints`	LIST_PATH, INTERSECTION_PATH, UNION_PATH, DIFFERENCE_PATH, SUBTRACTION_PATH

mine	the base URL
token	the API token

Author(s)

InterMine Team

See Also

ListManager-methods, ListManager-class

Examples

```
#hashed out example as requires API key
#im.human.list <- list_manager(initInterMine(listMines()["HumanMine"],"TOKEN")) #Add your API Token key
#class(im.human.list)
#resp <- get_list(im.human.list, " PL_Pax6_Targets")
#resp
```

---

newQuery *Initialize a new list query*

Description

A query needs to have at least view, constraints and constraintLogic. The view is a vector containing the columns of the query output. The constraint is a matrix containing the following columns: path (the path of the constraint), op (the constraint operator, one of '=', '!=', 'LOOKUP', 'ONE OF', 'NONE OF', '>', '<', '>=', '<=', 'LIKE'), value (the constraint value), code (the name of the constraint), extraValue (optional, required for LOOKUP constraints). The constraintLogic by default is "AND" operation, e.g., "A and B", where A and B are the codes in the constraints.

Usage

```
newQuery(name="", view=character(), sortOrder="", longDescription="",
      constraintLogic=NULL)
```

Arguments

name	a string, representing the name of the query.
view	a character vector, representing the fields to be selected from InterMine.
sortOrder	a string, representing the field according to which the query result is sorted and the sort order ("asc" or "desc"), following the format "FIELD ORDER".
longDescription	a string, representing the description of the query.
constraintLogic	a string, representing the logical relationship between the constraints, e.g., "A or B" where "A" and "B" are the codes in the constraints.

Value

a list representing the query.

Author(s)

InterMine Team

See Also

runQuery

Examples

```
nq <-  newQuery()
```

PL_DiabetesGenes *PL_DiabetesGenes data*

Description

A dataset containing identifiers of genes associated with all forms of Diabetes according to OMIM http://www.omim.org/.

Usage

```
data("PL_DiabetesGenes")
```

Format

A data frame with 68 observations on the following 6 variables:

Gene.symbol Gene symbol

Gene.name Gene whole name

Gene.primaryIdentifier InterMine Gene.primaryIdentifier (ENTREZ identifier)

Gene.secondaryIdentifier InterMine Gene.secondaryIdentifier (ENSEMBLE identifier)

Gene.length Gene length in base pairs

Gene.organism.name Gene organism name

Source

http://www.humanmine.org/humanmine/bag.do?subtab=view

Examples

```
data(PL_DiabetesGenes)
```

PL_FlyTF_site_specific_TFs *PL_FlyTF_site_specific_TFs data*

Description

A dataset containing identifiers of transcription factors with experimental evidence for both DNA-binding and transcriptional regulatory function. This data comes from www.flyTF.org, the drosophila transcription factor database version 2.

Usage

```
data("PL_FlyTF_site_specific_TFs")
```

Format

A data frame with 171 observations on the following 5 variables:

Index Index of observations

Gene.secondaryIdentifier InterMine Gene.secondaryIdentifier

Gene.symbol Gene symbol

Gene.primaryIdentifier InterMine Gene.primaryIdentifier

Gene.organism.name Gene organism name

Source

http://www.flymine.org/flymine/bag.do?subtab=view

Examples

```
data(PL_FlyTF_site_specific_TFs)
```

runQuery *Run InterMineR queries*

Description

Returns results from a query against data held inside the mine. These queries are similar to SQL queries, in that they request certain defined output columns of output, filtering the results through a series of "constraints".

Usage

```
runQuery(im, qry, timeout = 60)
```

Arguments

im	a Service object containing the base URL and API token, created by initInterMine.
qry	an InterMineR or a list object, representing the query to the database.
timeout	an integer, representing the number of seconds to wait for the web service to respond.

Details

Use setQuery function to create an InterMineR object. For setting a single constraint with multiple values, the function setConstraints can be used.

Alternatively, the user can define manually the constraints, the selection of data to be returned and the value by which they are ordered, as a list object.

For more information checkout the vignette of the package.

Value

a data.frame containing the data which were retrieved from the InterMine instance.

Methods

S4 method for class 'InterMineR'

The method accepts an object of the class InterMineR and uses its information to perform the query on the defined InterMine instance.

InterMineR-class objects can contain a single constraint with multiple values.

S4 method for class 'list'

runQuery(im, qry = "InterMineR") runQuery(im, qry = "list") The method accepts an object of the class list and uses its information to perform the query on the defined InterMine instance.

Queries defined as lists can possess only constraints with one value.

Author(s)

InterMine Team

See Also

setConstraints, setQuery, InterMineR-class, newQuery

Examples

```
# 1. Adapt 'GO_Gene' template query from FlyMine to 'InterMineR'

# get FlyMine instance
im.fly = initInterMine(listMines()["FlyMine"])

# get GO_Gene template query
qGO_Gene = getTemplateQuery(im.fly, "GO_Gene")

# constraint with GO value
```

```
qGO_Gene$where[[3]]

# modify GO_Gene template query to have more than one GO values
go.constraints = setConstraints(
  values = list(c("DNA repair", "cellular response to DNA damage stimulus")),
  modifyQueryConstraints = qGO_Gene,
  m.index = 3
)

go.constraints[[3]]

# create 'InterMineR' object
go.query = setQuery(
  inheritQuery = qGO_Gene,
  where = go.constraints
)

go.query

# run InterMineR query
go.results = runQuery(
  im = im.fly,
  qry = go.query
)

head(go.results)

# 2. Create similar query manually for Homo sapiens, using HumanMine

# get HumanMine instance
im.human = initInterMine(listMines()["HumanMine"])

# create constraints using GO terms and organism as values
hsa.go.constraints = setConstraints(
  paths = c("Gene.goAnnotation.ontologyTerm.parents.name",
            "Gene.organism.name"),
  operators = rep("=", 2),
  values = list(c("DNA repair", "cellular response to DNA damage stimulus"),
                  "Homo sapiens")
)

hsa.go.constraints

# create 'InterMineR' object
hsa.go.query = setQuery(
  select = c("Gene.secondaryIdentifier",
             "Gene.symbol",
             "Gene.goAnnotation.ontologyTerm.parents.name",
             "Gene.goAnnotation.ontologyTerm.parents.identifier",
             "Gene.goAnnotation.ontologyTerm.name",
             "Gene.goAnnotation.ontologyTerm.identifier"),
  orderBy = list(c(Gene.secondaryIdentifier = "ASC")),
  where = hsa.go.constraints
```

```
)

hsa.go.query

# run InterMineR query
hsa.go.results = runQuery(
  im = im.human,
  qry = hsa.go.query
)

head(hsa.go.results)
```

`Service-class` *Service class represents connections to different InterMine WebServices*

Description

The Service class is the main interface for the user. It will provide access to queries and templates, as well as doing the background task of fetching the data model, and actually requesting the query results.

Creating Objects

Objects can be created using the function `initInterMine`.

Slots

mine the URL base.

Author(s)

InterMine Team

See Also

`InterMineR`, `webservice-class`, `ListManager-methods`

`setConstraints` *setConstraints function is used to create a new or modify an existing list of constraints*

Description

setConstraints function is used to create or modify a list containing the constraints for an InterMine query. These constraints can be later passed on to a query formed by `setQuery` function.

Usage

```
setConstraints(
  paths,
  operators,
  values,
  modifyQueryConstraints,
  m.index
)
```

Arguments

`paths`	a character vector defining in which bioterm type the value(s) of each constraint belong. One assigned to each constraint.
`operators`	a character vector describing the operators for each constraint.
`values`	a list containing the values assigned to each constraints. The path of each argument defines the type of value by which the results of the query will be filtered.
`modifyQueryConstraints`	an existing list query (e.g. template query) whose constraints will be modified.
`m.index`	a vector of integer values indicating which constraints of the list assigned to modifyQueryConstraints will be modified.

Details

It is important to note that:

1. setConstraints assigns each argument to a constraint based on the order in which they are assigned and one at a time. This means that paths, operators and values arguments must be of the same length and carefully assigned.

2. setConstraints can create a list of new constraints or modify an existing one with the arguments modifyQueryConstraints and m.index

3. only one of the constraints can be assigned with multiple values, aiming to retrieve results for multiple bioterms (genes, etc.)

4. to assign multiple bioterms in a constraint, values argument is assigned with a vector at the appropriate order: e.g.

to assign c("a", "b") to the second constraint one must use either:

values = list(first_element, c("a", "b"), ...)

or when modifying an existing list of constraints:

values = list(c("a", "b"))

m.index = 2

Value

a list containing the constraints for an InterMineR query. It can be passed to setQuery function.

Author(s)

InterMine Team

See Also

setQuery, runQuery, InterMineR-class

Examples

```
# get mine instance
im.fly = initInterMine(listMines()["FlyMine"])

# get GO_Gene template query
qGO_Gene = getTemplateQuery(im.fly, "GO_Gene")

# constraint with GO value
qGO_Gene$where[[3]]

# modify GO_Gene template query to have more than one GO values
go.constraints = setConstraints(
  values = list(c("DNA repair", "cellular response to DNA damage stimulus")),
  modifyQueryConstraints = qGO_Gene,
  m.index = 3
)

go.constraints[[3]]

# 2. Create similar query manually for Homo sapiens, using HumanMine

# get HumanMine instance
im.human = initInterMine(listMines()["HumanMine"])

# create constraints using GO terms and organism as values
hsa.go.constraints = setConstraints(
  paths = c("Gene.goAnnotation.ontologyTerm.parents.name",
            "Gene.organism.name"),
  operators = rep("=", 2),
  values = list(c("DNA repair", "cellular response to DNA damage stimulus"),
                  "Homo sapiens")
)
```

```
hsa.go.constraints
```

`setQuery` *Initialize a new InterMineR query or modify an existing list query*

Description

setQuery function is used to create a query for an InterMine instance. It can also accept and modify pre-defined queries of class list, like the template queries which can be retrieved with the function `getTemplateQuery`. It returns a defined query stored as an object of class 'InterMineR'.

A query needs to have at least view, constraints and constraintLogic. The view is a vector containing the columns of the query output. The constraint is a matrix containing the following columns: path (the path of the constraint), op (the constraint operator, one of '=', '!=', 'LOOKUP', 'ONE OF', 'NONE OF', '>', '<', '>=', '<=', 'LIKE'), value (the constraint value), code (the name of the constraint), extraValue (optional, required for LOOKUP constraints). The constraintLogic by default is "AND" operation, e.g., "A and B", where A and B are the codes in the constraints.

Usage

```
setQuery(
  select,
  orderBy,
  where,
  name = "",
  description = "",
  inheritQuery
)
```

Arguments

`select`	a character vector defining the type of data to be returned.
`orderBy`	a list the name of the column and the type of sorting which will be used to order the retrieved data.frame.
`where`	a list containing the constraints used to restrict the query. Each constraint constitutes a list as well. If one wishes for a single constraint to possess more than one values, the `setConstraints` function can be used.
`name`	Assign with a character string giving a name to the query. Pre-fixed with "".
`description`	Assign with a character string describing the purpose and the type of data retrieved by the query. Pre-fixed with "".
`inheritQuery`	Assign with a pre-defined query list, the input values of which can be modified and inherited in the new query object of class 'InterMineR'.

Details

setQuery function can be used to create queries with a single constraint containing multiple values. The constraints can be first defined with the setConstraints function. The resulting 'InterMineR' object is passed to the runQuery function to perform the query.

Value

An InterMineR object

Author(s)

InterMine Team

See Also

setConstraints, runQuery, InterMineR-class

Examples

```
# get mine instance
im.fly = initInterMine(listMines()["FlyMine"])

# get GO_Gene template query
qGO_Gene = getTemplateQuery(im.fly, "GO_Gene")

# constraint with GO value
qGO_Gene$where[[3]]

# modify GO_Gene template query to have more than one GO values
go.constraints = setConstraints(
  values = list(c("DNA repair", "cellular response to DNA damage stimulus")),
  modifyQueryConstraints = qGO_Gene,
  m.index = 3
)
go.constraints[[3]]

# create 'InterMineR' object
go.query = setQuery(
  inheritQuery = qGO_Gene,
  where = go.constraints
)

class(go.query)
go.query

# 2. Create similar query manually for Homo sapiens, using HumanMine

# get HumanMine instance
im.human = initInterMine(listMines()["HumanMine"])

# create constraints using GO terms and organism as values
hsa.go.constraints = setConstraints(
```

```
  paths = c("Gene.goAnnotation.ontologyTerm.parents.name",
            "Gene.organism.name"),
  operators = rep("=", 2),
  values = list(c("DNA repair", "cellular response to DNA damage stimulus"),
                "Homo sapiens")
)

hsa.go.constraints

# create 'InterMineR' object
hsa.go.query = setQuery(
  select = c("Gene.secondaryIdentifier",
             "Gene.symbol",
             "Gene.goAnnotation.ontologyTerm.parents.name",
             "Gene.goAnnotation.ontologyTerm.parents.identifier",
             "Gene.goAnnotation.ontologyTerm.name",
             "Gene.goAnnotation.ontologyTerm.identifier"),
  orderBy = list(c(Gene.secondaryIdentifier = "ASC")),
  where = hsa.go.constraints
)

class(hsa.go.query)
hsa.go.query
```

`simplifyResult` *Convert multiple values of a column into Comma-separated character strings*

Description

This function converts the values of a column variable within a dataset into comma-separated, character strings. The process is achieved by using another column of the same dataset as index.

Usage

```
simplifyResult(
  dataset,
  index_column,
  values_column,
  returnInDataframe = FALSE
)
```

Arguments

`dataset`	a data.frame containing the data
`index_column`	a character string or a numeric/integer value indicating the column which will be used as index.

values_column a character string or a numeric/integer value indicating the column whose values will be converted into comma-separated, character strings.

returnInDataframe a logical value indicating whether to return only the converted columns or to append the character strings to the dataset.

Value

If returnInDataframe argument is set to FALSE then a data.frame is returned with the unique values of the index column and the character strings of the values_column that correspond to each.

If returnInDataframe argument is set to TRUE then the original dataset is returned containing an extra column in which the character strings of the values_column have been appended.

Author(s)

InterMine Team

Examples

```
# get HumanMine
im.human = initInterMine(listMines()["HumanMine"])

# get template query for retrieving GOTerms for specific genes
qGene_GO = getTemplateQuery(im.human, "Gene_GO")

# retrieve GOTerms for four different genes
rGene_GO = list(NULL)
for(i in seq(length(c("PPARG", "RPL5", "RPL11", "TP53")))){

  g = c("PPARG", "RPL5", "RPL11", "TP53")[i]
  qGene_GO$where[[1]]$value = g
  rGene_GO[[i]] = runQuery(im.human, qGene_GO)

}

# rbind results to data.frame
rGene_GO = do.call(rbind, rGene_GO)

# return simplified GOTerms results for each Gene
simplify_GOTerms = simplifyResult(
  dataset = rGene_GO,
  index_column = "Gene.symbol",
  values_column = "Gene.goAnnotation.ontologyTerm.identifier",
  returnInDataframe = FALSE
)

# return simplified GOTerms results for each Gene, within the original data.frame
simplify_GOTerms.2 = simplifyResult(
  dataset = rGene_GO,
  index_column = "Gene.symbol",
  values_column = "Gene.goAnnotation.ontologyTerm.identifier",
```

```
  returnInDataframe = TRUE
)
```

---

summary *Summarize InterMineR query constraints*

Description

Summarize the information about the constraints contained by an object of the class InterMineR.

Usage

```
summary(object,...)
```

Arguments

object	an object of the class InterMineR.
...	additional arguments affecting the summary produced.

Value

a data.frame containing the constraints of the InterMineR object as rows. Each constraint is constituted by a path, an operator and one or more values. Multiple values are returned as a comma-separated character string.

Author(s)

InterMine Team

See Also

InterMineR-class, setQuery

Examples

```
# get FlyMine instance
im.fly = initInterMine(listMines()["FlyMine"])

# get GO_Gene template query
qGO_Gene = getTemplateQuery(im.fly, "GO_Gene")

# modify GO_Gene template query to have more than one GO values
go.constraints = setConstraints(
  values = list(c("DNA repair", "cellular response to DNA damage stimulus")),
  modifyQueryConstraints = qGO_Gene,
  m.index = 3
)
```

```
# create 'InterMineR' object
go.query = setQuery(
  inheritQuery = qGO_Gene,
  where = go.constraints
)

# get InterMineR constraint summary
summary(go.query)
```

Index

∗ **datasets**
PL_DiabetesGenes, 31
PL_FlyTF_site_specific_TFs, 32

convertToGeneAnswers, 3
convertToGRanges, 5
convertToRangedSummarizedExperiment, *7*, 8
create_list *(*ListManager-methods*)*, 26
create_list,ListManager-method *(*ListManager-methods*)*, 26
create_list-methods *(*ListManager-methods*)*, 26

delete_lists *(*ListManager-methods*)*, 26
delete_lists,ListManager-method *(*ListManager-methods*)*, 26
delete_lists-methods *(*ListManager-methods*)*, 26
difference *(*ListManager-methods*)*, 26
difference,ListManager-method *(*ListManager-methods*)*, 26
difference-methods *(*ListManager-methods*)*, 26
doEnrichment, *3*, *4*, 10, *15*, *16*, *21*
doEnrichment,webservice-methods *(*doEnrichment*)*, 10
doEnrichment-methods *(*doEnrichment*)*, 10

GeneAnswers, *3*
get_list *(*ListManager-methods*)*, 26
get_list,ListManager-method *(*ListManager-methods*)*, 26
get_list-methods *(*ListManager-methods*)*, 26
getDatasets, 13, *24*
getDescription *(*InterMineR-methods*)*, 23
getDescription,InterMineR-method *(*InterMineR-methods*)*, 23
getDescription-methods *(*InterMineR-methods*)*, 23
getGeneIds, *11*, *12*, 15, *21*
getModel, *4*, *13*, *14*, 16
getName *(*InterMineR-methods*)*, 23
getName,InterMineR-method *(*InterMineR-methods*)*, 23
getName-methods *(*InterMineR-methods*)*, 23
getOrderBy *(*InterMineR-methods*)*, 23
getOrderBy,InterMineR-method *(*InterMineR-methods*)*, 23
getOrderBy-methods *(*InterMineR-methods*)*, 23
getRelease, 17
getSelect *(*InterMineR-methods*)*, 23
getSelect,InterMineR-method *(*InterMineR-methods*)*, 23
getSelect-methods *(*InterMineR-methods*)*, 23
getTemplateQuery, 18, *38*
getTemplates, 19
getVersion, 20
getWhere *(*InterMineR-methods*)*, 23
getWhere,InterMineR-method *(*InterMineR-methods*)*, 23
getWhere-methods *(*InterMineR-methods*)*, 23
getWidgets, *4*, *11*, *12*, *16*, 20
GRanges, *5–7*

initInterMine, *8*, *13*, *15–20*, 21, *24*, *33*
InterMineR, *35*
InterMineR *(*InterMineR-package*)*, 3
InterMineR-class, 22, *23*
InterMineR-methods, 23
InterMineR-package, 3
intersect *(*ListManager-methods*)*, 26
intersect,ListManager-method *(*ListManager-methods*)*, 26

intersect-methods
(ListManager-methods), 26

list_manager, *25, 26*, 29
list_manager,webservice-methods
(list_manager), 29
list_manager-methods (list_manager), 29
listDatasets, *13, 14*, 24
ListManager-class, 25, *26*
ListManager-methods, 26
listMines, 28

newQuery, 30, *33*

PL_DiabetesGenes, 31
PL_FlyTF_site_specific_TFs, 32

RangedSummarizedExperiment, *8, 9*
reshape, *9*
runQuery, *26*, *31*, 32, *37*, *39*
runQuery,ANY,InterMineR-method
(runQuery), 32
runQuery,ANY,list-method (runQuery), 32

Service-class, 35
setConstraints, *23*, *33*, 36, *38, 39*
setQuery, *22, 23*, *33*, *36, 37*, 38, *42*
simplifyResult, 40
subtract (ListManager-methods), 26
subtract,ListManager-method
(ListManager-methods), 26
subtract-methods (ListManager-methods),
26
SummarizedExperiment, *9*
summary, 42
summary,ANY-method (summary), 42
summary,InterMineR-method (summary), 42

union (ListManager-methods), 26
union,ListManager-method
(ListManager-methods), 26
union-methods (ListManager-methods), 26

webservice-class (Service-class), 35
webservice-methods (doEnrichment), 10
webservice-methods (list_manager), 29