Package 'tranSMART'

July 2, 2012

Type Package

Title tranSMART API

Version 1.0

Date 2012-07-02

Author mmcduffie

Maintainer mmcduffie@recomdata.com

Description This package contains function which act as an API for retrieving data from the tranSMART data warehouse.

License Apache 2.0

LazyLoad yes

R topics documented:

`tranSMART-package` *Functionality for interacting with the tranSMART data warehouse.*

Description

Functions that prepare SQL queries that run against the tranSMART data warehouse. Some extra functionality exists to transform the data after querying and before rendering to user.

Details

Package:	tranSMART
Type:	Package
Version:	1.0
Date:	2012-04-13

Author(s)

mmcduffie

Maintainer: mmcduffie <mmcduffie@recomdata.com>

Examples

```
ls("package:tranSMART")

lsf.str("package:tranSMART")
```

`tranSMART.DB.connection`
This variable holds the connection object to the oracle database.

Description

This object is initialized before every command, creating the connection to the database.

Usage

```
tranSMART.DB.connection <- tranSMART.DB.establishConnection()
```

Examples

```
##Establish a connection to a database
##(Assuming the server,username and password variables have been created.)
tranSMART.DB.connection <- tranSMART.DB.establishConnection()
```

tranSMART.DB.dbname

A character representing the URL of the database to connect to.

Description

When the tranSMART packages tries to connect to the database, this will be used in the connection string.

Usage

```
data(tranSMART.DB.dbname)
```

Format

The format is: chr "//ADDRESS:PORT/SID"

Examples

```
tranSMART.DB.username <- 'someuser'
tranSMART.DB.password <- 'somepassword'
tranSMART.DB.dbname <- 'somedb'

tranSMART.DB.connection <- tranSMART.DB.establishConnection()
```

tranSMART.DB.establishConnection

Function to establish the connection to the tranSMART DB.

Description

Using tranSMART.DB.username,tranSMART.DB.password,tranSMART.DB.dbname a connection to the tranSMART data warehouse will be established and returned.

Usage

```
tranSMART.DB.establishConnection()
```

Examples

```
tranSMART.DB.username <- 'someuser'
tranSMART.DB.password <- 'somepassword'
tranSMART.DB.dbname <- 'somedb'

tranSMART.DB.connection <- tranSMART.DB.establishConnection()
```

tranSMART.DB.password

Password for the tranSMART database login.

Description

When the tranSMART packages tries to connect to the database, this will be used in the connection string.

Usage

```
data(tranSMART.DB.password)
```

Format

The format is: chr "somedb_password"

Examples

```
tranSMART.DB.username <- 'someuser'
tranSMART.DB.password <- 'somepassword'
tranSMART.DB.dbname <- 'somedb'

tranSMART.DB.connection <- tranSMART.DB.establishConnection()
```

tranSMART.DB.username

Username for the tranSMART database login.

Usage

```
data(tranSMART.DB.username)
```

Format

The format is: chr "somedb_user"

Examples

```
tranSMART.DB.username <- 'someuser'
tranSMART.DB.password <- 'somepassword'
tranSMART.DB.dbname <- 'somedb'

tranSMART.DB.connection <- tranSMART.DB.establishConnection()
```

`transmart.getClinicalData` *This function returns Clinical Data from the tranSMART database.*

Description

Clinical Data (aka low dimensional data) represents clinical parameters that have been collected about patients/samples. These things could include medical history, demographics, or low complexity assays. In order to retrieve this data the user must supply a list of codes which can be retrieved using the getDistinctConcepts function.

Usage

```
transmart.getClinicalData(concepts.codelist,
data.pivot = TRUE,
concepts.prePivotTrim = TRUE,
concepts.trimLengths = 4,
sql.print = FALSE)
```

Arguments

`concepts.codelist`
: A list of concept cds. This would ideally be passed in from the return of the getDistinctConcepts function.

`data.pivot`
: A boolean indicating whether the data should be pivoted after retrieving.

`concepts.prePivotTrim`
: This boolean indicates if the concept paths should be trimmed before the pivot occurs. A concept path is a "\" seperated string indicating a concepts relation to other concepts. The path is laid out visually in the dataset explorer tree within tranSMART. By trimming items off before pivoting columns can be collapsed. An example is when multiple studies worth of data are retrieved but the path is trimmed so that the survival time node from all 4 studies has the same concept code. When the data is pivoted all the surival times will be put into one column.

`concepts.trimLengths`
: This is the number of "\" seperated elements to remove from the concept path when trimming. A negative trim length will leave items from the end of the path while positive lengths remove from the begining.

`sql.print`
: Not used as this time.

Value

A data frame is returned with a patient num (internal identifier), Subject ID (ID that is taken from the raw input files when data is loaded into tranSMART), Trial Name (Name of the study) and the remaining columns are the clinical data items. If the data is not pivoted, it will be represented in a long format instead of a wide one.

Examples

```
#Assumes a connection to the tranSMART DB has been made.

conceptList <- transmart.getDistinctConcepts(studyList = c('GSE5287'),
pathMatchList = c('%Overall%'))
clinicalData <- transmart.getClinicalData(concepts.codelist = conceptList$CONCEPT_CD,
concepts.trimLengths = -1)
```

---

transmart.getClinicalMutationData

This functions performs a very specific search for a Mutation Type concept within the ontology tree.

Description

In order to pull a list of the mutations that a patient has a search is done on the ontology to find the concept that represents the mutations on the genes specified. A data frame is constructed with has a column per supplied gene and a row per patient in the studies supplied. The intersection lists the concept (if one exists).

Usage

```
transmart.getClinicalMutationData(study.list,gene.list,trimLength = 4)
```

Arguments

`study.list`	A list of studies to limit the text search to. This lookup is case insensitive.
`gene.list`	A list of genes to look for mutations for.
`trimLength`	This is the length of the resulting concept code at a patient/gene intersection.

Value

A data frame is returned with PATIENT_ID (Internal Identifier), SUBJECT_ID (Identifier from source data), STUDY ID (Internal ID) and a column for each gene specified. In each column is the result of searching to see if that patient was associated with a mutation in that gene. If NA is present that means the concept was not associated with this patient (ie. they do not have the mutation).

Examples

```
mutData <- transmart.getClinicalMutationData(
study.list = c('VELCADE024','VELCADE025'),
gene.list=c('NRAS','BRAF','MET'),
trimLength = 4)
```

`transmart.getDistinctConcepts`
This function finds clinical concepts in the tranSMART database that match the supplied strings.

Description

Before clinical data can be retrieved from the tranSMART warehouse the user must find the concept codes associated with the data to be retrieved. This function will perform a search of the concept paths within tranSMART, restricted to a list of studies.

Usage

```
transmart.getDistinctConcepts(studyList = NULL, pathMatchList)
```

Arguments

`studyList`	A list of studies to limit the text search to. This lookup is case insensitive. If this is not supplied the results will be for all studies.
`pathMatchList`	A list of strings to search on. The wildcard character will be placed before and after each term when looking for a matching concept path. This search term is case insensitive.

Value

A data frame is returned with a column for the concept code (Which will be used in the getClinicalData function), the concept path and a count of the number of patients who have that concept. A column with the study ID is also supplied.

Examples

```
#Assumes a connection to the tranSMART DB has been made.

conceptList <- transmart.getDistinctConcepts(studyList = c('GSE20685'),
pathMatchList = c('duration'))
clinicalData <- transmart.getClinicalData(concepts.codelist = conceptList$CONCEPT_D)
```

`transmart.getGeneGoMembership`
Retrieve GeneGo pathway information from the tranSMART database.

Description

This function will take a pathway name and return all the genes in that pathway. Optionally the user can download all the pathway information by leaving the parameter blank.

Usage

```
transmart.getGeneGoMembership(genegoName = NA)
```

Arguments

genegoName	The name of the GeneGo pathway to look up genes for.

Examples

```
allResults <- transmart.getGeneGoMembership()
byPathway <- transmart.getGeneGoMembership(
genegoName = c('alanine, cysteine, and L-methionine metabolism'))
```

---

transmart.getGEXData

This function gets Gene Expression data from the tranSMART database.

Description

This function will retrieve Gene Expression data based on filters. Options are available to pivot and aggregate the data.

Usage

```
transmart.getGEXData(study.list = NA,
gene.list = NA,
pathway = NA,
signature = NA,
patient.list = NA,
sample.type.list = NA,
tissue.type.list = NA,
timepoint.list = NA,
platform.list = NULL,
probe.list = NULL,
platform.removeOnOverlap = NULL,
show.genes = FALSE,
print.statement = FALSE,
data.pivot = TRUE,
data.pivot.aggregate = NA)
```

Arguments

study.list	A list of studies to limit the text search to. This lookup is case insensitive.
gene.list	A list of genes to filter the GEX by.
pathway	A pathway (Exact text match, case insensitive.)
signature	The name of a gene signature that was previously loaded into the tranSMART database.

`patient.list` A list of patient IDs to filter the data by. This is the patient identifier generated within tranSMART.

`sample.type.list`
A list of sample types to filter the data by. The available sample types can be retrieved using the transmart.listHDDAttributes function.

`tissue.type.list`
A list of tissue types to filter the data by. The available tissue types can be retrieved using the transmart.listHDDAttributes function.

`timepoint.list`
A list of timepoints to filter the data by. The available timepoints can be retrieved using the transmart.listHDDAttributes function.

`platform.list`
This is a list of platforms to filter on.

`probe.list` A list of probe ids to filter the results by.

`platform.removeOnOverlap`
This is a list of platforms that you want to be *overlapped* if there are probes in multiple platforms. The script will check for probes in multiple platforms and remove the records that have a platform in the provided list.

`show.genes` If this is set to true a gene column will be added to data output.

`print.statement`
If this is set to true the function will only print the SQL statement to retrieve the GEX data instead of running it.

`data.pivot` Flag indicating whether the GEX data should be pivoted or not.

`data.pivot.aggregate`
Function to use when aggregating data during a pivot. Use this only if you know what you are aggregating.

`data.pivot.patient_id`
This will use the tranSMART internal identifier which is guarenteed to be unique per study (barring any data issues).

Examples

```
#---
#Getting Gene Expression data by Study and a list of genes.
gexData <- transmart.getGEXData(study.list = c('Gse10021'),
gene.list=c('BrCA2'))
#---

#---
#Getting Gene Expression data by study and a signature loaded into the
#tranSMART data warehouse.
gexData <- transmart.getGEXData(study.list = c('GSE10021'),
signature=c('Breast Cancer Lung Metastasis'))
#---

#---
#Getting Gene Expression data by study and gene and using an aggregation function when
#pivoting. This is useful when there are multiple values per probe as in the case of
#some studies. If this function is not provided for those studies, a message warning
#the user about "length" being used will display and the returned data frame will
#almost certain contain funny looking data.
gexData <- transmart.getGEXData(study.list = c('Gse10021'),
```

```
gene.list=c('BrCA2'),
data.pivot.aggregate = median)
#---

#---
#Using get distinct concepts to generate a study list of public studies
#to get Gene Expression data for.
breastCancerStudies <- transmart.getDistinctConcepts(
pathMatchList = c('\\Public Studies\\%\\Cancer\\Breast Cancer\\%Affymetrix'))

#Pass in the list of studies, a list of genes,
#a function to aggregate the data with multiple records,
#a parameter to narrow down which platform to pull data for,
#and a boolean that will force the pivot to use
#our own unique internal ID as the patient ID.
breastCancerData <- transmart.getGEXData(
study.list=breastCancerStudies$STUDYCODE,
gene.list=c('BRCA1','BRCA2'),
platform.list = c('GPL96'),
data.pivot.aggregate = median,
data.pivot.patient_id = TRUE)
#---

#---
#Pull gene expression data for a study, for a given signature,
#with a platform filter but don't pivot it.
gep.tmp.oneplat=transmart.getGEXData(study.list="GSE10021",
signature="u133p2.gene.names",
platform.list = c('GPL96'),
data.pivot = FALSE)
#---

#---
#Pull gene expression data but remove any probes
#from GPL97 if they overlap with GPL96.
gep.tmp.overlapremoved=transmart.getGEXData(study.list="GSE10021",
platform.removeOnOverlap = c('GPL97'))
#---

#---
#Pull gene expression by a probe id.
gep.tmp.probelist = transmart.getGEXData(study.list="GSE10021",
probe.list=c('201172_x_at','202068_s_at'))
#---

#---
#Pull gene expression based on Sample Types.

#This gets a list of all public studies.
publicStudyList <- transmart.listStudies('GSE%')

#This gets the attributes for all those studies.
attr=transmart.listHDDAttributes(studyList = publicStudyList$STUDYCODE)

#Pull the Sample type into its own frame.
sampleTypes <- data.frame(attr[1])
```

```
#Pull all the BRCA2 GEX data for the Fifth SAMPLE_TYPE.
gexData <- transmart.getGEXData(study.list = publicStudyList$STUDYCODE,
sample.type.list = sampleTypes$SAMPLE_TYPE[5],
gene.list=c('BrCA2'))
#---
```

`transmart.getPatientMapping`

This function gets a data frame that contains a mapping between tranSMART unique ID's and the Subject IDs supplied in the curation process.

Description

tranSMART assigns unique IDs to patient records in addition to the identifier that is carried over from the data loading process. This function will get the mapping for a list of studies.

Usage

```
transmart.getPatientMapping(studyList)
```

Arguments

`studyList`	A list of studies to get the patient IDs for. Patient IDs are unique across studies where the ID provided in the data loading process may not be.

Examples

```
patientMappingFrame <- transmart.getPatientMapping(studyList = c('Gse10021'))
```

`transmart.getProbeGeneMapping`

This function will get a lookup table of probes/genes from the tranSMART database.

Description

tranSMART has annotation information stored that can be queried by either the probe or gene id to get the appropriate mapping information between the two.

Usage

```
transmart.getProbeGeneMapping(probeIds)
```

Arguments

`probeIds`	A list of probe identifiers to find the gene symbols for.
`geneIds`	A list of genes to look up the probe ids for.

Examples

```
transmart.getProbeGeneMapping(probeIds = c('220665_at','220675_s_at','220730_at'))
transmart.getProbeGeneMapping(geneIds = c('LUZP4'))
```

---

transmart.getProbeGeneSNPMapping

This function will get a lookup table of SNP Names/genes from the tranSMART database.

Description

tranSMART has annotation information stored that can be queried by either the SNP name or gene id to get the appropriate mapping information between the two.

Usage

```
transmart.getProbeGeneSNPMapping(probeIds = NA,geneIds = NA)
```

Arguments

probeIds	A list of SNP identifiers to find the gene symbols for.
geneIds	A list of genes to look up the SNP names for.

Examples

```
byProbe <- transmart.getProbeGeneSNPMapping(probeIds = c('SNP_A-1855402'))
byGene <- transmart.getProbeGeneSNPMapping(geneIds = c('MAPT'))
```

---

transmart.getSNPData

This function gets SNP data from the tranSMART database.

Description

This function will retrieve SNP data based on filters. Options are available to pivot and aggregate the data.

Usage

```
transmart.getSNPData(study.list = NA,
gene.list = NA,
pathway = NA,
signature = NA,
patient.list = NA,
sample.type.list = NA,
tissue.type.list = NA,
timepoint.list = NA,
platform.list = NULL,
probe.list = NULL,
```

```
platform.removeOnOverlap = NULL,
show.genes = FALSE,
print.statement = FALSE,
data.pivot = TRUE,
data.CN.pivot.aggregate = NA,
data.GT.pivot.aggregate = NA,
data.pivot.patient_id = FALSE)
```

Arguments

study.list	A list of studies to limit the text search to. This lookup is case insensitive.
gene.list	A list of genes to filter the SNP by.
pathway	A pathway (Exact text match, case insensitive.)
signature	The name of a gene signature that was previously loaded into the tranSMART database.
patient.list	A list of patient IDs to filter the data by. This is the patient identifier generated within tranSMART.
sample.type.list	A list of sample types to filter the data by. The available sample types can be retrieved using another function.
tissue.type.list	A list of tissue types to filter the data by. The available tissue types can be retrieved using another function.
timepoint.list	A list of timepoints to filter the data by. The available timepoints can be retrieved using another function.
platform.list	This is a list of platforms to filter on.
probe.list	A list of probe ids to filter the results by.
show.genes	If this is set to true a gene column will be added to data output.
print.statement	If this is set to true the function will only print the SQL statement to retrieve the SNP data instead of running it.
data.pivot	Flag indicating whether the SNP data should be pivoted or not.
data.CN.pivot.aggregate	Function to use when aggregating data during a pivot. Use this only if you know what you are aggregating.
data.GT.pivot.aggregate	Function to use when aggregating data during a pivot. Use this only if you know what you are aggregating.
data.pivot.patient_id	This will use the tranSMART internal identifier which is guarenteed to be unique per study (barring any data issues).

Value

If unpivoted, a single data frame is returned. If pivoted, a list of data frames is returned, one for Copy Number and one for Genotype.

DF1	Copy Number Data
DF2	Genotype Data

Examples

```
#Getting SNP Data for two studies, only probes in one gene, not pivoting the data.
unpivotSNP <- transmart.getSNPData(study.list = c('ONCOTEST_001','ONCOTEST_002'),
gene.list=c('BRCA2'),data.pivot = FALSE)

#Getting SNP Data by a list of genes for one study.
byGeneSNP <- transmart.getSNPData(study.list = c('ONCOTEST_001'),
gene.list=c('BRCA2','BRCA1'))

#Getting SNP data by a signature for one study.
bySigSNP <- transmart.getSNPData(study.list = c('ONCOTEST_001'),
signature='u133p2.gene.names')

#Getting SNP Data by a signature for multiples studies, forcing the pivot
#to use our internal ID.
bySigSNP <- transmart.getSNPData(study.list = c('ONCOTEST_001','ONCOTEST_002'),
signature='u133p2.gene.names',
data.pivot.patient_id = TRUE)

#Getting SNP data by a pathway.
byPath <- transmart.getSNPData(study.list = c('ONCOTEST_001'),pathway='synapsis')

#Getting SNP data by a list of probes.
byProbe <- transmart.getSNPData(study.list = c('ONCOTEST_001'),
probe.list=c('SNP_A-1827941','SNP_A-4203012','SNP_A-1869849'))

#Getting SNP data by probes, using our internal ID.
byProbePatient <- transmart.getSNPData(study.list = c('ONCOTEST_001'),
probe.list=c('SNP_A-1827941','SNP_A-4203012','SNP_A-1869849'),
data.pivot.patient_id = TRUE)

#Getting SNP data with a gene column.
bySigSNPFullParameters <- transmart.getSNPData(study.list = c('ONCOTEST_001'),
signature='u133p2.gene.names',show.genes = TRUE)

#Example of pulling two data frames from list.
CNV <- data.frame(byProbePatient[1])
GENO <- data.frame(byProbePatient[2])
```

---

transmart.listHDDAttributes

List High Dimensional Data Attributes within tranSMART.

Description

For a given list of studies this function will query the table that maps patients to samples. Returned is a list of 3 attribute types and their distinct values. These values can be used to filter in other functions, like retrieving gene expression data by a sample type.

Usage

```
transmart.listHDDAttributes(studyList)
```

Arguments

studyList	A list of studies to retrieve the mapping data for.

Value

A list of 3 data frames, each frame is a distinct list of a sample attribute type.

Examples

```
#---
#Pull gene expression based on Sample Types.

#This gets a list of all public studies.
publicStudyList <- transmart.listStudies('GSE%')

#This gets the attributes for all those studies.
attr=transmart.listHDDAttributes(studyList = publicStudyList$STUDYCODE)

#Pull the Sample type into its own frame.
sampleTypes <- data.frame(attr[1])

#Pull all the BRCA2 GEX data for the Fifth SAMPLE_TYPE.
gexData <- transmart.getGEXData(study.list = publicStudyList$STUDYCODE,
sample.type.list = sampleTypes$SAMPLE_TYPE[5],
gene.list=c('BrCA2'))
#---
```

---

transmart.listStudies

This function will list studies that have a character match with the supplied strings.

Description

This function uses the passed in search string (With % as wildcards) to generate a list of studies and their concept codes. The STUDYCODE column can be used to identify studies within the other R tranSMART functions. The second parameter is used to trim the concept path to collapse records.

Usage

```
transmart.listStudies(studyLike,concept.size = 4, gexFlag = FALSE)
```

Arguments

studyLike	Text string to search for.
concept.size	The number of "chunks" to keep in the concept path, the number actually reflects the number of "\" delimeters to keep. Default is 4.
gexFlag	If used this flag will return a count from the microarray table of distinct probe ids for the given study. This will indicate if GEX data is available in the study.

Value

A data.frame is returned with a STUDYCODE and CONCEPT_PATH column. Both of these come from the table that is used to generate the tree within the Dataset Explorer. The concepts are grouped by the Study Code and the Concept Path. You can pass the STUDYCODE column to the other tranSMART R functions as a study filter.

Examples

```
transmart.listStudies('GSE1079%',concept.trim = 5)
```

Index