Package ‘ontoProc’

March 23, 2024

Title  processing of ontologies of anatomy, cell lines, and so on

Description  Support harvesting of diverse bioinformatic ontologies, making particular use of the ontologyIndex package on CRAN. We provide snapshots of key ontologies for terms about cells, cell lines, chemical compounds, and anatomy, to help analyze genome-scale experiments, particularly cell x compound screens. Another purpose is to strengthen development of compelling use cases for richer interfaces to emerging ontologies.

Version  1.24.0

Imports  Biobase, S4Vectors, methods, stats, utils, BiocFileCache, shiny, graph, Rgraphviz, ontologyPlot, dplyr, magrittr, DT, igraph, AnnotationHub, SummarizedExperiment

Suggests  knitr, org.Hs.eg.db, org.Mm.eg.db, testthat, BiocStyle, SingleCellExperiment, celldex, rmarkdown, AnnotationDbi

Depends  R (>= 4.0), ontologyIndex

License  Artistic-2.0

LazyLoad  yes

biocViews  Infrastructure, GO

RoxygenNote  7.2.3

VignetteBuilder  knitr

Encoding  UTF-8


URL  https://github.com/vjcitn/ontoProc

BugReports  https://github.com/vjcitn/ontoProc/issues

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

git_branch  RELEASE_3_18
**R topics documented:**

- allGOterms ......................................................... 3
- bind_formal_tags .................................................. 3
- c.TermSet-method ................................................... 4
- cellTypeToGO ...................................................... 5
- cleanCLOTnames ................................................... 6
- CLfeats .......................................................... 6
- common_classes .................................................... 7
- connect_classes .................................................... 8
- ctmarks .......................................................... 8
- cyclicSigset ...................................................... 9
- demoApp .......................................................... 10
- dropStop .......................................................... 10
- fastGrep .......................................................... 11
- findCommonAncestors ............................................. 12
- getChebiLite ....................................................... 13
- getLeavesFromTerm ................................................ 14
- getOnto .......................................................... 15
- humrna ........................................................... 15
- improveNodes ..................................................... 16
- ldfToTerms ......................................................... 16
- liberalMap ......................................................... 18
- makeSelectInput .................................................. 18
- make_graphNEL_from_ontology_plot .............................. 19
- map2prose ........................................................ 20
- mapOneNaive ..................................................... 21
- minicorpus ....................................................... 21
- nomenCheckup ..................................................... 22
- onto_plot2 ........................................................ 23
- onto_roots ........................................................ 23
- packDesc2019 ..................................................... 24
- packDesc2021 ..................................................... 24
- packDesc2022 ..................................................... 25
- packDesc2023 ..................................................... 25
- PROSYM .......................................................... 26
- recognizedPredicates ............................................. 27
- secLevGen ........................................................ 27
allGOterms

allGOterms: data.frame with ids and terms

Description

allGOterms: data.frame with ids and terms

Usage

allGOterms

Format

data.frame instance

Source

This is a snapshot of all the terms available from GO.db (3.4.2), August 2017, using keys(GO.db, keytype="TERM").

Examples

data(allGOterms)
head(allGOterms)

bind_formal_tags

add mapping from informal to formal cell type tags to a SummarizedExperiment colData

Description

add mapping from informal to formal cell type tags to a SummarizedExperiment colData

Usage

bind_formal_tags(se, informal, tagmap, force = FALSE)
Arguments

se SummarizedExperiment instance
informal character(1) name of colData element with uncontrolled vocabulary
tagmap data.frame with columns 'informal' and 'formal'
force logical(1), defaults to FALSE; if TRUE, allows clobbering existing colData variable named "formal"

Value

SummarizedExperiment instance with a new colData column 'label.ont' giving the formal tags associated with each sample

Note

This function will fail if the value of 'informal' is not among the colData variable names, or if "formal" is among the colData variable names.

c,TermSet-method

Description

combine TermSet instances

Usage

## S4 method for signature 'TermSet'
c(x, ...)

Arguments

x TermSet instance
... additional instances

Value

TermSet instance
**cellTypeToGO**

utilities for approximate matching of cell type terms to GO categories and annotations

---

**Description**

utilities for approximate matching of cell type terms to GO categories and annotations

**Usage**

```r
cellTypeToGO(celltypeString, gotab, ...)
cellTypeToGenes(
  celltypeString,
  gotab,
  orgDb,
  cols = c("ENSEMBL", "SYMBOl"),
  ...
)
```

**Arguments**

- `celltypeString` character atom to be used to search GO terms using
- `gotab` a data.frame with columns GO (goids) and TERM (term strings) `agrep`
- `...` additional arguments to `agrep`
- `orgDb` instances of orgDb
- `cols` columns to be retrieved in select operation

**Value**

data.frame
data.frame

**Note**

Very primitive, uses agrep to try to find relevant terms.

**Examples**

```r
library(org.Hs.eg.db)
data(allGOTerms)
head(cellTypeToGO("serotonergic neuron", allGOTerms))
head(cellTypeToGenes("serotonergic neuron", allGOTerms, org.Hs.eg.db))
```
cleanCLOnames

obtain named character vector of terms from Cell Line Ontology, omitting obsolete and trailing 'cell'

**Description**

obtain named character vector of terms from Cell Line Ontology, omitting obsolete and trailing 'cell'

**Usage**

```r
cleanCLOnames()
```

**Value**

character()

**Examples**

```r
cleanCLOnames()[1:10]
```

---

CLfeats

produce a data.frame of features relevant to a Cell Ontology class

**Description**

produce a data.frame of features relevant to a Cell Ontology class

**Usage**

```r
CLfeats(ont, tag = "CL:0001054", pr, go)
```

**Arguments**

- **ont**: instance of ontologyIndex ontology
- **tag**: character(1) a CL: class tag
- **pr**: instance of ontologyIndex PRO protein ontology
- **go**: instance of ontologyIndex GO gene ontology

**Value**

a data.frame instance
Note

This function will look in the intersection_of and has_part, lacks_part components of the CL entry to find properties asserted of or inherited by the cell type identified in ‘tag’. As of 1.19, this function does not look in global environment for ontologies. We use 2021 versions in the examples because some changes in ontologies omit important relationships; revisions to package code after 1.19.4 will attempt to address these.

Examples

```r
c1 = getOnto("cellOnto", year_added="2021")
pr = getOnto("Pronto", "2021")  # legacy tag, for 2022 would be PROnto
go = getOnto("goOnto", "2021")
CLfeats(c1, tag="CL:0001054", pr=pr, go=go)
```

**common_classes**

<table>
<thead>
<tr>
<th>common_classes</th>
<th>list and count samples with common ontological annotation in two SEs</th>
</tr>
</thead>
</table>

Description

list and count samples with common ontological annotation in two SEs

Usage

```r
common_classes(ont, se1, se2)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ont</td>
<td>instance of ontologyIndex ontology</td>
</tr>
<tr>
<td>se1</td>
<td>a SummarizedExperiment using ‘label.ont‘ in colData to provide ontological tags (from ‘ont’) for samples</td>
</tr>
<tr>
<td>se2</td>
<td>a SummarizedExperiment using ‘label.ont‘ in colData to provide ontological tags (from ‘ont’) for samples</td>
</tr>
</tbody>
</table>

Value

a data.frame with rownames given by the common tags, the class names as column ‘clname’, and counts of samples bearing the given tags in remaining columns.

Examples

```r
if (requireNamespace("celldex")) {
  imm = celldex::ImmGenData()
  if (!"label.ont" %in% names(colData(imm))) {
    cl = getOnto("cellOnto")
    blu = celldex::BlueprintEncodeData()
    common_classes( cl, imm, blu )
  }
}
```
connect_classes

connect ontological categories between related, annotated SummarizedExperiments

Usage
connect_classes(ont, se1, se2)

Arguments
- ont: an ontologyIndex ontology instance
- se1: SummarizedExperiment instance with 'label.ont' among colData columns
- se2: SummarizedExperiment instance with 'label.ont' among colData columns

Value
a list with two sublists mapping from terms in one SE to descendant terms in the other SE

temarks

app to review molecular properties of cell types via cell ontology

Description
app to review molecular properties of cell types via cell ontology

Usage
temarks(cl, pr, go)

Arguments
- cl: an import of a Cell Ontology (or extended Cell Ontology) in ontology_index form
- pr: an import of a Protein Ontology in ontology_index form
- go: an import of a Gene Ontology in ontology_index form

Value
a data.frame with features for selected cell types
cyclicSigset

Note

Prototype of harvesting of cell ontology by searching has_part, has_plasma_membrane_part, intersection_of and allied ontology relationships. Uses shiny. Can perform better if getPROnto() and getGeneOnto() values are in .GlobalEnv as pr and go respectively.

Examples

```r
if (interactive()) {
  co = getOnto("cellOnto", year_added="2023") # has plasma membrane relations
  go = getOnto("goOnto", "2023")
  pr = getOnto("Pronto", "2021") # peculiar tag used in legacy, would be PROnto with 2022
  ctmrk(co, go, pr)
}
```

Description

as in Bakken et al. (2017 PMID 29322913) create gene signatures for k cell types, each of which fails to express all but one gene in a set of k genes

Usage

```
cyclicSigset(
  idvec,
  conds = c("hasExp", "lacksExp"),
  tags = paste0("CL:X", 1:length(idvec))
)
```

Arguments

- `idvec` character vector of identifiers, must have names() set to identify cells bearing genes
- `conds` character(2) tokens used to indicate condition to which signature element contributes
- `tags` character vector of cell-type identifiers; for Cell Ontology use CL: as prefix, one element for each element of idvec

Value

a long data.frame
Examples

```r
digels = c("CL:X01"="GRK3", "CL:X02"="NTNG1", "CL:X03"="BAGE2", "CL:X04"="MC4R", "CL:X05"="PAX6", "CL:X06"="TSPAN12", "CL:X07"="hSHISA8", "CL:X08"="NNGG", "CL:X09"="ARHGEF28", "CL:X10"="EGF")
sigdf = cyclicSigset(sigels)
head(sigdf)
```

demoApp  
*demonstrate the use of makeSelectInput*

Description

demonstrate the use of makeSelectInput

Usage

demoApp()

Value

Run only for side effect of starting a shiny app.

Examples

```r
if (interactive()) {
  require(shiny)
  print(demoApp())
}
```

dropStop  
*dropStop is a utility for removing certain words from text data*

Description

dropStop is a utility for removing certain words from text data

Usage

dropStop(x, drop, lower = TRUE, splitby = " ")

Arguments

- `x`: character vector of strings to be cleaned
- `drop`: character vector of words to scrub
- `lower`: logical, if TRUE, x converted with `tolower`
- `splitby`: character, used with `strsplit` to tokenize x
Value

a list with one element per input string, split by " ", with elements in drop removed

Examples

```r
data(minicorpus)
minicorpus[1:3]
dropStop(minicorpus)[1:3]
```

---

fastGrep

some fields of interest are lists, and grep per se should not be used – this function checks and uses grep within vapply when appropriate

Description

some fields of interest are lists, and grep per se should not be used – this function checks and uses grep within vapply when appropriate

Usage

```r
fastGrep(patt, onto, field, ...)
```

Arguments

- `patt` a regular expression whose presence in field should be checked
- `onto` an ontologyIndex instance
- `field` the ontologyIndex component to be searched
- `...` passed to grep

Value

logical vector indicating vector or list elements where a match is found

Examples

```r
cheb = getOnto("chebi_lite")
ind = fastGrep("tanespimycin", cheb, "name")
cheb$name[ind]
```
Description
Given a set of ontology terms, find their latest common ancestors based on the term hierarchy.

Usage
findCommonAncestors(..., g, remove.self = TRUE, descriptions = NULL)

Arguments
... One or more (possibly named) character vectors containing ontology terms.
g A graph object containing the hierarchy of all ontology terms.
remove.self Logical scalar indicating whether to ignore ancestors containing only a single term (themselves).
descriptions Named character vector containing plain-English descriptions for each term. Names should be the term identifier while the values are the descriptions.

Details
This function identifies all terms in g that are the latest common ancestor (LCA) of any subset of terms in ... An LCA is one that has no children that have the exact same set of descendent terms in ..., i.e., it is the most specific term for that set of observed descendents. Knowing the LCA is useful for deciding how terms should be rolled up to broader definitions in downstream applications, usually when the exact terms in ... are too specific for practical use.

The descendents DataFrame in each row of the output describes the descendents for each LCA, stratified by their presence or absence in each entry of ... This is particularly useful for seeing how different sets of terms would be aggregated into broader terms, e.g., when harmonizing annotation from different datasets or studies. Note that any names for ... will be reflected in the columns of the DataFrame for each LCA.

Value
A DataFrame where each row corresponds to a common ancestor term. This contains the columns number, the number of descendent terms across all vectors in ...; and descendents, a List of DataFrames containing the identities of the descendents. It may also contain the column description, containing the description for each term.

Author(s)
Aaron Lun
Examples

co <- getOnto("cellOnto")

# TODO: wrap in utility function.
parents <- co$parents
self <- rep(names(parents), lengths(parents))
library(igraph)
g <- make_graph(rbind(unlist(parents), self))

# Selecting random terms:
LCA <- ontoProc:::findCommonAncestors(A=sample(names(V(g)), 20),
  B=sample(names(V(g)), 20), g=g)

LCA[1,]
LCA[1,"descendents"][1]

getChebiLite  basic getters in old style, retained 2023 for deprecation interval

Description

basic getters in old style, retained 2023 for deprecation interval

Usage

getchebiLite()
getcellosaurusOnto()
getricerontNE()
getchebiOnto()
getoNTreeOnto()
gediseaseOnto()
geneOnto()
getHCAOnto()
getPROnto()
getPATOnto()
getMondoOnto()
getSIOnto()
getLeavesFromTerm

**Value**

instance of ontology_index (S3) from ontologyIndex

**Note**

getChebiOnto loads ontoRda/chebi_full.rda
getOncotreeOnto loads ontoRda/oncotree.rda
getDiseaseOnto loads ontoRda/diseaseOnto.rda
getHCAOnto loads ontoRda/hcaOnto.rda produced from hcao.owl at https://github.com/HumanCellAtlas/ontology/releases/tag/1.0.6, 2/11/2019, python pronto was used to convert OWL to OBO.
getPROnto loads ontoRda/PRonto.rda, produced from http://purl.obolibrary.org/obo/pr.obo 'reasoned' ontology from OBO foundry, 02-08-2019. In contrast to other ontologies, this is imported via get_OBO with ‘extract_tags=’minimal’.

---

getLeavesFromTerm obtain childless descendents of a term (including query)

**Description**

obtain childless descendents of a term (including query)

**Usage**

getLeavesFromTerm(x, ont)

**Arguments**

x a character(1) id element for ontology_index instance

ont an ontology_index instance as defined in ontologyIndex package

**Value**

character vector of 'leaves' of ontology tree

**Examples**

ch = getOnto("chebi_lite")
alldr = getLeavesFromTerm("CHEBI:23888", ch)
head(ch$name[alldr[1:15]])
getOnto

**get the ontology based on a short tag and year**

**Description**

get the ontology based on a short tag and year

**Usage**

```r
getOnto(ontoname = "cellOnto", year_added = "2023")
```

**Arguments**

- **ontoname** character(1) must be an element in `valid_ontonames()`
- **year_added** character(1) refers to `rdatadateadded` in AnnotationHub metadata

**Note**

This queries AnnotationHub for "ontoProcData" and then filters to find the AnnotationHub accession number and retrieves the ontologyIndex serialization of the associated OBO representation of the ontology.

**Examples**

```r
c = getOnto()
tail(c$name[1000:1500])
```

---

**humrna**

**humrna: a data.frame of SRA metadata related to RNA-seq in humans**

**Description**

humrna: a data.frame of SRA metadata related to RNA-seq in humans

**Usage**

```r
humrna
```

**Format**

data.frame

**Note**

arbitrarily chosen from RNA-seq studies for taxon 9606
Source

NCBI SRA

Examples

data(humrna)
names(humrna)
head(humrna[,1:5])

importNodes
inject linefeeds for node names for graph, with textual annotation from ontology

Description

inject linefeeds for node names for graph, with textual annotation from ontology

Usage

importNodes(g, ont)

Arguments

g  graphNEL instance
ont  instance of ontology from ontologyIndex

ldfToTerms
use output of cyclicSigset to generate a series of character vectors constituting OBO terms

Description

use output of cyclicSigset to generate a series of character vectors constituting OBO terms

Usage

ldfToTerms(
  ldf,
  propmap,
  sigels,
  prologMaker = function(id, ...) sprintf("id: %s", id)
)
ldfToTerms

Arguments

- **ldf**: a 'long format' data.frame as created by cyclicSigset
- **propmap**: a character vector with names of elements corresponding to 'abbreviated' relationship tokens and element values corresponding to full relationship-naming strings
- **sigels**: a named character vector associating cell types (names) to genes expressed in a cyclic set, one element per type
- **prologMaker**: a function with arguments (id, ...), in which id is character(1), that generates a vector of strings that will be used for each cell type-specific term.

Value

a character vector, strings can be concatenated to OBO

Note

ldfToTerms is not sufficiently general to produce terms for any reasonably populated long data frame/propmap combination, but it is a working example for the cyclic set context.

Examples

```r
# a set of cell types -- names are cell type token, values are genes expressed in a cyclic set -- each cell type expresses exactly one gene in the set and fails to express all the other genes in the set. See Figs 3 and 4 of Bakken et al [PMID 29322913].
sigels = c("CL:X01"="GRIK3", "CL:X02"="NTNG1", "CL:X03"="BAGE2", "CL:X04"="MC4R", "CL:X05"="PAX6", "CL:X06"="TSPAN12", "CL:X07"="hSHISA8", "CL:X08"="SNCG", "CL:X09"="ARHGEF28", "CL:X10"="EGF")

# create the associated long data frame
ldf = cyclicSigset(sigels)

# describe the abbreviations
pmap = c("hasExp"="has_expression_of", "lacksExp"="lacks_expression_of")

# now define the prolog for each cell type
makeIntnProlog = function(id, ...) {
  # make type-specific prologs as key-value pairs
  c(
    sprintf("id: %s", id),
    sprintf("name: %s-expressing cortical layer 1 interneuron, human", ...),
    sprintf("def: '%s-expressing cortical layer 1 interneuron, human described via RNA-seq observations' [PMID 29322913]; is_a: CL:0000099 ! interneuron; intersection_of: CL:0000099 ! interneuron")
  )
}
tms = ldfToTerms(ldf, pmap, sigels, makeIntnProlog)
cat(tms[[1]], sep="\n")
```
liberalMap

Produce a data.frame with a set of naive terms mapped to all matching ontology ids and their formal terms.

Description

Produce a data.frame with a set of naive terms mapped to all matching ontology ids and their formal terms.

Usage

liberalMap(terms, onto, useAgrep = FALSE, ...)

Arguments

terms character() vector, can use grep-compatible regular expressions
onto an instance of ontologyIndex::ontology_index
useAgrep logical(1) if TRUE, agrep will be used
... passed to agrep if used

Value

a data.frame

Examples

cands = c("astrocyte$", "oligodendrocyte", "oligodendrocyte precursor", "neoplastic", "^neuron$", "^vascular", "badterm")
#co = ontoProc::getCellOnto()
co = getOnto("cellOnto", year_added="2023")
liberalMap(cands, co)

makeSelectInput

generate a selectInput control for an ontologyIndex slice

Description

generate a selectInput control for an ontologyIndex slice
Usage

```r
makeSelectInput(
  onto,
  term,
  type = "siblings",
  inputId,
  label,
  multiple = TRUE,
  ...
)
```

Arguments

- `onto` ontologyIndex instance
- `term` character(1) term used as basis for term list option set in the control
- `type` character(1) 'siblings' or 'children', relationship to 'term' that the options will satisfy
- `inputId` character(1) for use in server
- `label` character(1) for labeling in ui
- `multiple` logical(1) passed to `selectInput`
- ... additional parameters passed to `selectInput`

Value

a `selectInput` control

Examples

```r
makeSelectInput
```

---

`make_graphNEL_from_ontology_plot`

obtain graphNEL from `ontology_plot` instance of `ontologyPlot`

---

Description

obtain graphNEL from `ontology_plot` instance of `ontologyPlot`

Usage

```r
make_graphNEL_from_ontology_plot(x)
```

Arguments

- `x` instance of S3 class `ontology_plot`
Value

instance of S4 graphNEL class

Examples

```r
requireNamespace("Rgraphviz")
requireNamespace("graph")
cl = getOnto("cellOnto")
        "CL:0000576", "CL:0000623", "CL:0000451", "CL:0000556")
p3k = ontologyPlot::onto_plot(cl, c13k)
gnel = make_graphNEL_from_ontology_plot(p3k)
gnel = improveNodes(gnel, cl)
graph::graph.par(list(nodes=list(shape="plaintext", cex=.8)))
gnel = Rgraphviz::layoutGraph(gnel)
Rgraphviz::renderGraph(gnel)
```

Description

use prose terminology with output of connect_classes

Usage

```r
map2prose(x, cl)
```

Arguments

- `x`: a component of connect_classes output
- `cl`: an ontologyIndex ontology instance

Value

a decorated list
mapOneNaive

use grep or agrep to find a match for a naive token into ontology

Description

use grep or agrep to find a match for a naive token into ontology

Usage

mapOneNaive(naive, onto, useAgrep = FALSE, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>naive</td>
<td>character(1)</td>
</tr>
<tr>
<td>onto</td>
<td>an instance of ontologyIndex::ontology_index</td>
</tr>
<tr>
<td>useAgrep</td>
<td>logical(1) if TRUE, agrep will be used</td>
</tr>
<tr>
<td>...</td>
<td>passed to agrep if used</td>
</tr>
</tbody>
</table>

Value

if a match is found, the result of grep/agrep with value=TRUE is returned; otherwise a named NA_character_ is returned
	named vector, names are ontology identifiers, values are matched strings

Examples

#co = ontoProc::getCellOnto()
co = getOnto("cellOnto", year_added="2023")
mapOneNaive("astrocyte", co)

minicorpus

minicorpus: a vector of annotation strings found in 'study title' of SRA metadata.

Description

minicorpus: a vector of annotation strings found in 'study title' of SRA metadata.

Usage

minicorpus

Format

character vector
nomenCheckup

Note

arbitrarily chosen from titles of RNA-seq studies for taxon 9606

Source

NCBI SRA

Examples

data(minicorpus)
head(minicorpus)

nomenCheckup

repair nomenclature mismatches (to curated term set) in a vector of terms

Description

repair nomenclature mismatches (to curated term set) in a vector of terms

Usage

nomenCheckup(cand, namedOffic, n = 1, tagcolname = "tag", ...)

Arguments
cand character vector of candidate terms
namedOffic named character vector of curated terms, the names are regarded as tags, intended to be identifiers in curated ontologies
n numeric(1) number of nearest neighbors to return
tagcolname character(1) prefix used to name columns for tags in output
... passed to adist

Value

a data.frame instance with 2n+1 columns (column 1 is candidate, remaining n pairs of columns are (term, tag) for n nearest neighbors as measured by adist.

Examples

candidates = c("JHH7", "HUT102", "HS739T", "NCIH716")
# the candidates are cell line names returned in the text dump from
# https://portals.broadinstitute.org/ccle/page?gene=AHR
# note that one must travel to the third nearest neighbor
# to find the match (and tag) for Hs 739.T
# in this example, we compare to cell line names in Cell Line Ontology
nomenCheckup(candidates, cleanCLOnames(), n=3, tagcolname="clo")
onto_plot2

high-level use of graph/Rgraphviz for rendering ontology relations

Description

high-level use of graph/Rgraphviz for rendering ontology relations

Usage

onto_plot2(ont, terms2use, cex = 0.8, ...)

Arguments

ont instance of ontology from ontologyIndex

terms2use character vector

cex numeric(1) defaults to .8, supplied to Rgraphviz::graph.par

... passed to onto_plot of ontologyPlot

Value

graphNEL instance (invisibly)

Examples

c1 = getOnto("cellOnto")
"CL:0000576", "CL:0000623", "CL:0000451", "CL:0000556")
onto_plot2(c1, c13k)

onto_roots

list parentless nodes in ontology_index instance

Description

list parentless nodes in ontology_index instance

Usage

onto_roots(x)

Arguments

x an ontology_index instance
Value

a report (produced by cat()) of root ids and associated names

Examples

onto_roots

---

packDesc2019

Description

packDesc2019: overview of ontoProc resources

Usage

packDesc2019

Format

data.frame instance

Note

Brief survey of functions available to load serialized ontology_index instances imported from OBO.

Examples

data(packDesc2019)
head(packDesc2019)

---

packDesc2021

Description

packDesc2021: overview of ontoProc resources

Usage

packDesc2021

Format

data.frame instance
**Note**

Brief survey of functions available to load serialized ontology_index instances imported from OBO. Focus is on versions added in 2021.

**Examples**

```r
data(packDesc2021)
head(packDesc2021)
```

---

**packDesc2022**

*packDesc2022: overview of ontoProc resources*

**Description**

packDesc2022: overview of ontoProc resources

**Usage**

packDesc2022

**Format**

data.frame instance

**Note**

Brief survey of functions available to load serialized ontology_index instances imported from OBO. Focus is on versions added in 2022.

**Examples**

```r
data(packDesc2022)
head(packDesc2022)
```

---

**packDesc2023**

*packDesc2023: overview of ontoProc resources*

**Description**

packDesc2023: overview of ontoProc resources

**Usage**

packDesc2023
PROSYM

Format

data.frame instance

Note

Brief survey of functions available to load serialized ontology_index instances imported from OBO. Focus is on versions added in 2023. Several manual interventions were needed – cellosaurus was too large to use the script in inst/scripts/desc.R, and a number of ontologies do not have 2023 versions.

Examples

data(packDesc2023)
head(packDesc2023)

---

PROSYM

PROSYM: HGNC symbol synonyms for PR (protein ontology) entries identified in Cell Ontology

Description

PROSYM: HGNC symbol synonyms for PR (protein ontology) entries identified in Cell Ontology

Usage

PROSYM

Format

data.frame instance

Note

This is a snapshot of the synonyms component of an extract_tags='everything' import of PR. The 'EXACT.*PRO-short.*:DNx' pattern is used to retrieve HGNC symbols. See ?getPROnto for more provenance information.

Source

OBO Foundry

Examples

data(PROSYM)
head(PROSYM)
recognizedPredicates  enumerate ontological relationships used in ontoProc utilities

Description

enumerate ontological relationships used in ontoProc utilities

Usage

recognizedPredicates()

Value

character vector, names of elements are abbreviated tokens that may be used in code

Examples

head(recognizedPredicates())

secLevGen  simple generation of children of 'choices' given as terms, returned as TermSet

Description

simple generation of children of 'choices' given as terms, returned as TermSet

Usage

secLevGen(choices, ont)

Arguments

choices  vector of terms
ont  instance of ontology_index (S3) from ontologyIndex package

Value

TermSet instance

Examples

efoOnto = getOnto("efoOnto")
secLevGen( "disease", efoOnto )
selectFromMap

select a set of elements from a term 'map' and return a contribution to a data.frame

Description

select a set of elements from a term 'map' and return a contribution to a data.frame

Usage

selectFromMap(namedvec, index)

Arguments

namedvec named character vector, as returned from mapOneNaive
index numeric() or integer(), typically of length one

Value

a data.frame; if index does not inherit from numeric, a data.frame of one row with columns 'ontoid' and 'term' populated with NA_character_ is returned, otherwise a similarly named data.frame is returned with contents from the selected elements of namedvec

Examples

#co = ontoProc::getCellOnto()
co = getOnto("cellOnto", year_added="2023")
mast = mapOneNaive("astrocyte", co)
selectFromMap(mast, 1)

seur3kTab

tabulate the basic outcome of PBMC 3K tutorial of Seurat

Description

tabulate the basic outcome of PBMC 3K tutorial of Seurat

Usage

seur3kTab()

Value

a data.frame

Examples

seur3kTab()
siblings_TAG

generate a TermSet with siblings of a given term, excluding that term by default

Description

generate a TermSet with siblings of a given term, excluding that term by default
acquire the label of an ontology subject tag
acquire the labels of children of an ontology subject tag

Usage

siblings_TAG(Tagstring = "EFO:1001209", ontology, justSibs = TRUE)

label_TAG(Tagstring = "EFO:0000311", ontology)

children_TAG(Tagstring = "EFO:1001209", ontology)

Arguments

Tagstring a character(1) that identifies a term
ontology instance of ontology_index (S3) from ontologyIndex
justSibs character(1)

Value

TermSet instance
character(1)
TermSet instance

Note

for label_TAG, Tagstring may be a vector

Examples

efoOnto = getOnto("efoOnto")
siblings_TAG( "EFO:1001209", efoOnto )
efoOnto = getOnto("efoOnto")
label_TAG( "EFO:0000311", efoOnto )
efoOnto = getOnto("efoOnto")
children_TAG( ontology = efoOnto )
stopWords

*stopWords: vector of stop words from xpo6.com*

**Description**

stopWords: vector of stop words from xpo6.com

**Usage**

stopWords

**Format**

character vector

**Note**

"Stop words" are English words that are assumed to contribute limited semantic value in the analysis of free text.

**Source**

http://xpo6.com/list-of-english-stop-words/

**Examples**

```r
data(stopWords)
head(stopWords)
```

subset_descendants

*subset a SummarizedExperiment to which ontology tags have been bound using 'bind_formal_tags', obtaining the 'descendants' of the class of interest*

**Description**

subset a SummarizedExperiment to which ontology tags have been bound using 'bind_formal_tags', obtaining the 'descendants' of the class of interest

**Usage**

```r
subset_descendants(
  se,
  onto,
  class_name,
  class_tag,
  formal_cd_name = "label.ont"
)
```
**sym2CellOnto**

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>se</td>
<td>SummarizedExperiment instance</td>
</tr>
<tr>
<td>onto</td>
<td>representation of an ontology using representation from ontologyIndex package</td>
</tr>
<tr>
<td>class_name</td>
<td>character(1) if 'class_tag' is missing, this will be grepped in onto[['name']] to find class and its descendants</td>
</tr>
<tr>
<td>class_tag</td>
<td>character(1) used if given to identify &quot;ontological descendants&quot; of this term in se</td>
</tr>
<tr>
<td>formal_cd_name</td>
<td>character(1) tells name used for ontology tag column in 'colData(se)'</td>
</tr>
</tbody>
</table>

**Value**

instance of SummarizedExperiment

**Description**

use Cell Ontology and Protein Ontology to identify cell-type defining conditions in which a given gene is named

**Usage**

sym2CellOnto(sym, cl, pr)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sym</td>
<td>gene symbol, must be used in protein ontology as a PRO:DNx exact match token</td>
</tr>
<tr>
<td>cl</td>
<td>result of getOnto(&quot;cellOnto&quot;)</td>
</tr>
<tr>
<td>pr</td>
<td>result of getOnto(&quot;PROnto&quot;)</td>
</tr>
</tbody>
</table>

**Value**

DataFrame if any hits are found. A field 'cond' abbreviates the identified conditions: (has/lacks)PMP (plasma membrane part) (hi/lo)PMAmt (plasma membrane amount), (has/lacks)Part.

**Note**

Currently just checks for *plasma_membrane_part, *plasma_membrane_amount, and *Part conditions.

**Examples**

if (!exists("cl")) cl = getOnto("cellOnto")
if (!exists("pr")) pr = getOnto("PROnto")
sym2CellOnto("ITGAM", cl, pr)
sym2CellOnto("FOXP3", cl, pr)
TermSet-class  

manage ontological data with tags and a DataFrame instance

abbreviated display for TermSet instances

Usage

## S4 method for signature 'TermSet'
show(object)

Arguments

object  instance of TermSet class

Value

instance of TermSet

Examples

efoOnto = getOnto("efoOnto")
defsibs = siblings_TAG("EFO:1001209", efoOnto)
class(defsibs)
defsibs

valid_ontonames  
give a vector of valid 'names' of ontoProc ontologies

Description

give a vector of valid 'names' of ontoProc ontologies

Usage

valid_ontonames()

Examples

head(valid_ontonames())
Index

**datasets**

- allGOterms, 3
- humrna, 15
- minicorpus, 21
- packDesc2019, 24
- packDesc2021, 24
- packDesc2022, 25
- packDesc2023, 25
- PROSYM, 26
- stopWords, 30

- adist, 22
- agrep, 5
- allGOterms, 3

- bind_formal_tags, 3

- c,TermSet-method, 4
- cellTypeToGenes (cellTypeToGO), 5
- cellTypeToGO, 5
- children_TAG (siblings_TAG), 29
- cleanCLOnames, 6
- CLfeats, 6
- common_classes, 7
- connect_classes, 8
- ctmarks, 8
- cyclicSigset, 9

- DataFrame, 12
- demoApp, 10
- dropStop, 10

- fastGrep, 11
- findCommonAncestors, 12

- getCellosaurusOnto (getChebiLite), 13
- getChebiLite, 13
- getChebiOnto (getChebiLite), 13
- getDiseaseOnto (getChebiLite), 13
- getGeneOnto (getChebiLite), 13
- getHCAOnto (getChebiLite), 13
- getLeavesFromTerm, 14
- getMondoOnto (getChebiLite), 13
- getOnctreeOnto (getChebiLite), 13
- getOnto, 15
- getPATOnto (getChebiLite), 13
- getPROnto (getChebiLite), 13
- getSIOOnto (getChebiLite), 13
- getUBERON_NE (getChebiLite), 13
- graph, 12

- humrna, 15

- improveNodes, 16

- label_TAG (siblings_TAG), 29
- ldfToTerms, 16
- liberalMap, 18
- List, 12

- make_graphNEL_from_ontology_plot, 19
- makeSelectInput, 18
- map2prose, 20
- mapOneNaive, 21, 28
- minicorpus, 21

- nomenCheckup, 22

- onto_plot2, 23
- onto_roots, 23

- packDesc2019, 24
- packDesc2021, 24
- packDesc2022, 25
- packDesc2023, 25
- PROSYM, 26

- recognizedPredicates, 27

- secLevGen, 27
- selectFromMap, 28
- selectInput, 79

33
seur3kTab, 28
show(TermSet-class), 32
show, TermSet-method (TermSet-class), 32
siblings_TAG, 29
stopWords, 30
subset_descendants, 30
sym2CellOnto, 31

TermSet-class, 32
tolower, 10

valid_ontonames, 32