**Package ‘ontoProc’**

May 18, 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**  1.26.0

**Imports**  Biobase, S4Vectors, methods, stats, utils, BiocFileCache, shiny, graph, Rgraphviz, ontologyPlot, dplyr, magrittr, DT, igraph, AnnotationHub, SummarizedExperiment, reticulate, R.utils, httr

**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**  7.3.1

**VignetteBuilder**  knitr

**Encoding**  UTF-8


**URL**  [https://github.com/vjcitn/ontoProc](https://github.com/vjcitn/ontoProc)

**BugReports**  [https://github.com/vjcitn/ontoProc/issues](https://github.com/vjcitn/ontoProc/issues)
Config/reticulate/autoconfigure
list( packages = list( list(package = ~ owlready2~ ) ) )

SystemRequirements
owlready2

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

git_branch RELEASE_3_19

git_last_commit 781f959

Repository Bioconductor 3.19

Date/Publication 2024-05-17

Author Vincent Carey [ctb, cre] (<https://orcid.org/0000-0003-4046-0063>), Sara Stankiewicz [ctb]

Maintainer Vincent Carey <stvjc@channing.harvard.edu>

Contents

allGOterms ................................................................. 3
ancestors ................................................................. 4
ancestors_names .......................................................... 5
bind_formal_tags .......................................................... 5
c.TermSet-method .......................................................... 6
cellTypeToGO ............................................................... 7
children_names ............................................................ 8
cleanCLOnames ............................................................. 8
CLfeats ................................................................. 9
common_classes ............................................................ 10
connect_classes ............................................................ 10
cmarks ................................................................. 11
cyclicSigset ............................................................... 12
demoApp ................................................................. 13
dropStop ................................................................. 13
fastGrep ................................................................. 14
findCommonAncestors ..................................................... 15
getcheblite ............................................................... 16
getLeavesFromTerm .......................................................... 17
getOnto ................................................................. 18
get_classes ............................................................... 18
get_ordo_owl_path ........................................................... 19
humrna ................................................................. 19
improveNodes .............................................................. 20
labels.owlents ............................................................. 20
ldfToTerms ............................................................... 21
liberalMap ............................................................... 22
makeSelectInput ............................................................ 23
make_graphNEL_from_ontology_plot .................................. 24
map2prose ............................................................. 24
allGOterms

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)

ancestors retrieve ancestor 'sets'

Description

retrieve ancestor 'sets'

Usage

ancestors(oe)

Arguments

oe owlents instance

Value

a list of sets

Examples

pa = get_ordo_owl_path()
orde = setup_entities(pa)
orde ancestors(orde[1:5])
labels(orde[1:5])
ancestors_names

obtain list of names of a set of ancestors

Description
obtain list of names of a set of ancestors

Usage
ancestors_names(anclist)

Arguments
anclist output of ‘ancestors’

Value
list of vectors of character()

Note
non-entities are removed and names are extracted

Examples
pa = get_ordo_owl_path()
orde = setup_entities(pa)
al = ancestors(orde[1001:1002])
ancestors_names(al)

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)
c.TermSet-method

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>se</code></td>
<td>SummarizedExperiment instance</td>
</tr>
<tr>
<td><code>informal</code></td>
<td>character(1) name of colData element with uncontrolled vocabulary</td>
</tr>
<tr>
<td><code>tagmap</code></td>
<td>data.frame with columns 'informal' and 'formal'</td>
</tr>
<tr>
<td><code>force</code></td>
<td>logical(1), defaults to FALSE; if TRUE, allows clobbering existing colData variable named &quot;formal&quot;</td>
</tr>
</tbody>
</table>

**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.

### Usage

```r
## S4 method for signature 'TermSet'
c(x, 
```

**Description**

combine TermSet instances

**Usage**

```r
## S4 method for signature 'TermSet'
c(x, 
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td>TermSet instance</td>
</tr>
<tr>
<td><code>...</code></td>
<td>additional instances</td>
</tr>
</tbody>
</table>

**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

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

library(org.Hs.eg.db)
data(allGOterms)
head(cellTypeToGO("serotonergic neuron", allGOterms))
head(cellTypeToGenes("serotonergic neuron", allGOterms, org.Hs.eg.db))
children_names  
*obtain list of names of a set of subclasses/children*

**Description**

obtain list of names of a set of subclasses/children

**Usage**

children_names(sclist)

**Arguments**

sclist  
output of 'subclasses'

**Value**

list of vectors of character()

**Note**

non-entities are removed and names are extracted

**Examples**

```r
pa = get_ordo_owl_path()
orde = setup_entities(pa)
al = subclasses(orde[100:120])
children_names(al)
```

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**

cleanCLOnames()

**Value**

character()
CLfeats

Examples

   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

   CLfeats(ont, tag = "CL:001054", 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

   cl = getOnto("cellOnto", year_added="2021")
   pr = getOnto("Pronto", "2021")  # legacy tag, for 2022 would be PROnto
   go = getOnto("goOnto", "2021")
   CLfeats(cl, tag="CL:001054", pr=pr, go=go)
common_classes

list and count samples with common ontological annotation in two SEs

Description

list and count samples with common ontological annotation in two SEs

Usage

common_classes(ont, se1, se2)

Arguments

ont instance of ontologyIndex ontology
se1 a SummarizedExperiment using ‘label.ont’ in colData to provide ontological tags (from ‘ont’) for samples
se2 a SummarizedExperiment using ‘label.ont’ in colData to provide ontological tags (from ‘ont’) for samples

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

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

connect_classes

connect ontological categories between related, annotated SummarizedExperiments

Description

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

---

cmarks

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

---

**Description**

app to review molecular properties of cell types via cell ontology

**Usage**

cmarks(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

**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
  cmarks(co, go, pr)
}
```
cyclicSigset

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

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
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")
sigdf = cyclicSigset(sigels)
head(sigdf)
```
**demoApp**

`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**

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**

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**

tcheb = getOnto("chebi_lite")
ind = fastGrep("tanespimycin", cheb, "name")
cheb$name[ind]
findCommonAncestors  

**Description**

Given a set of ontology terms, find their latest common ancestors based on the term hierarchy.

**Usage**

```r
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

c <- getOnto("cellOnto")

# TODO: wrap in utility function.
parents <- c$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**

gChebiLite()

cCellosaurusOnto()

cUBERON_NE()

cChebiOnto()

cOncotreeOnto()

cDiseaseOnto()

cGeneOnto()

cHCAOnto()

cPROnto()

cPATOnto()

cMondoOnto()

cSIOOnto()
**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**

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

get_classes

return a generator with ontology classes

Description
return a generator with ontology classes

Usage
get_classes(owlfile)

Arguments
owlfile reference to OWL file, can be URL, will be processed by owlready2.get_ontology

Value
generator with output of classes() on the loaded ontology
get_ordo_owl_path

decompress ordo owl file

Description
decompress ordo owl file

Usage
get_ordo_owl_path(target = tempdir())

Arguments
target character(1) path to where decompressed owl will live

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
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])
improveNodes

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

improveNodes(g, ont)

Arguments

g graphNEL instance
ont instance of ontology from ontologyIndex

labels.owlents

retrieve labels with names

Description

retrieve labels with names

Usage

## S3 method for class 'owlents'
labels(object, ...)

Arguments

object owlents instance
... not used

Note

When multiple labels are present, only first is silently returned. Note that reticulate 1.35.0 made a change that appears to imply that `[0]` can be used to retrieve the desired components. To get ontology tags, use `names(labels(...))`.
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

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

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 293...]
    "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
...

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

make_graphNEL_from_ontology_plot(x)

Arguments

x  
instance of S3 class ontology_plot

Value

instance of S4 graphNEL class

Examples

requireNamespace("Rgraphviz")
requireNamespace("graph")
cl = getOnto("cellOnto")
    "CL:000576", "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)

map2prose

use prose terminology with output of connect_classes

Description

use prose terminology with output of connect_classes

Usage

map2prose(x, cl)
**mapOneNaive**

**Arguments**

- `x` a component of connect_classes output
- `c1` 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**

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

**Arguments**

- `naive` character(1)
- `onto` an instance of ontologyIndex::ontology_index
- `useAgrep` logical(1) if TRUE, agrep will be used
- `...` passed to agrep if used

**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**

```r
#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

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", ...)
onto_plot2

**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

**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

**Value**
graphNEL instance (invisibly)
Examples

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

Example output:
```
onto_roots
```

Description

List parentless nodes in `ontology_index` instance.

Usage

```r
onto_roots(x)
```

Arguments

- `x`: An `ontology_index` instance

Value

A report (produced by `cat()` of root ids and associated names.

Examples

```r
onto_roots
```

owl2cache

**Description**

Cache an OWL file accessible via URL.

**Usage**

```r
owl2cache(cache = BiocFileCache::BiocFileCache(), url)
```

**Arguments**

- `cache`: BiocFileCache instance or equivalent
- `url`: character(1)
Note

This function will check for presence of url in cache using bfcquery; if a hit is found, returns the rpath associated with the last matching record. etags can be available for use with bfcneedsupdate.

Examples

```r
ca = BiocFileCache::BiocFileCache()
hppa = owl2cache(ca,
    url="http://purl.obolibrary.org/obo/hp/releases/2023-10-09/hp-base.owl")
setup_entities(hppa)
```

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

```r
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**

data(packDesc2021)
head(packDesc2021)

packDesc2022

**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**

data(packDesc2022)
head(packDesc2022)
packDesc2023

Description
packDesc2023: overview of ontoProc resources

Usage
packDesc2023

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)

parents retrieve is_a

Description
retrieve is_a

Usage
parents(oe)

Arguments
oe owlents instance

Value
list of vectors of tags of parents
Examples

```r
pa = get_ordo_owl_path()
orde = setup_entities(pa)
orde
parents(orde[1000:1001])
labels(orde[1000:1001])
```

plot.owlents visualize ontology selection via onto_plot2, based on owlents

Description

visualize ontology selection via onto_plot2, based on owlents

Usage

```r
plot.owlents(x, y, ..., dropThing = TRUE)
```

Arguments

- `x` owlents instance
- `y` character() vector of entries in x$clnames
- `...` passed to onto_plot2
- `dropThing` logical(1) defaults to TRUE; if "Thing" is present in terms to display, it is removed

Examples

```r
c13k = c("CL:0000492", "CL:0001054", "CL:000236",
          "CL:000625", "CL:000576",
          "CL:0000623", "CL:000451", "CL:0000556")
c13k = gsub("\:"_, ", c13k)
clont_path = owl2cache(url="http://purl.obolibrary.org/obo/cl.owl")
clont = setup_entities(clont_path)
plot(clont,c13k)
```
print.owents
short printer

Description
short printer

Usage
---
## S3 method for class 'owents'
print(x, ...)

Arguments
---
x owlents instance
... not used

PROSYM

| 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)

setup_entities

construct owlents instance from an owl file

Description

construct owlents instance from an owl file

Usage

setup_entities(owlfn)

Arguments

owlfn character(1) path to valid owl ontology
Value

instance of owlents, which is a list with clnames (a vector of term names in form `\[namespace\]_[tag]`), allents (a list with python references to owlready2 entities, that can be operated on using owlready2.EntityClass methods), owlfn (filename), iri (IRI), call (record of call producing the entity.)

Examples

```r
pa = get_ordo_owl_path()
orde = setup_entities(pa)
orde
ancestors(orde[1000:1001])
labels(orde[1000:1001])
```

---

seur3kTab

*tabulate the basic outcome of PBMC 3K tutorial of Seurat*

Description

*tabulate the basic outcome of PBMC 3K tutorial of Seurat*

Usage

```r
seur3kTab()
```

Value

a data.frame

Examples

```r
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**

```r
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

data(stopWords)
head(stopWords)

subclasses retrieve subclass entities

Description

retrieve subclass entities

Usage

subclasses(oe)

Arguments

oe owlents instance

Examples

pa = get_ordo_owl_path()
orde = setup Entities(pa)
orde
sc <- subclasses(orde[1:5])
labels(orde[3])
o3 = reticulate::iterate(sc[[3]])
print(length(o3))
o3[[2]]
labels(orde["Orphanet_100011"])
subset_descendants

Description

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

Usage

subset_descendants(
  se,
  onto,
  class_name,
  class_tag,
  formal_cd_name = "label.ont"
)

Arguments

se             SummarizedExperiment instance
onto           representation of an ontology using representation from ontologyIndex package
class_name     character(1) if 'class_tag' is missing, this will be grepped in onto["name"] to find class and its descendants
class_tag      character(1) used if given to identify "ontological descendants" of this term in se
formal_cd_name character(1) tells name used for ontology tag column in 'colData(se)'

Value

instance of SummarizedExperiment

sym2CellOnto

Description

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

```r
sym2CellOnto(sym, cl, pr)
```

**Arguments**

- `sym` gene symbol, must be used in protein ontology as a PRO:DNx exact match token
- `cl` result of `getOnto("cellOnto")`
- `pr` result of `getOnto("PROnto")`

**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**

```r
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

**Description**

manage ontological data with tags and a DataFrame instance

abbreviated display for TermSet instances

**Usage**

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

**Arguments**

- `object` instance of TermSet class

**Value**

instance of TermSet
Examples

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

**url_ok**

`url_ok(url)`

**Arguments**

`url` character(1)

**Value**

logical(1)

**valid_ononames**

`valid_ononames()`

**Description**

give a vector of valid 'names' of ontoProc ontologies

**Usage**

`valid_ononames()`

**Examples**

`head(valid_ononames())`
Description

subset method

Usage

```r
## S3 method for class 'owlents'
x[i, j, drop = FALSE]
```

Arguments

- `x`         owlents instance
- `i`         character or numeric vector
- `j`         not used
- `drop`      not used
Index

* datasets
  allGOterms, 3
  humrna, 19
  minicorpus, 26
  packDesc2019, 29
  packDesc2021, 30
  packDesc2022, 30
  packDesc2023, 31
  PROSYM, 33
  stopwords, 37
  [.owlents, 42

  adist, 27
  agrep, 7
  allGOterms, 3
  ancestors, 4
  ancestors_names, 5

bind_formal_tags, 5

c,TermSet-method, 6
getCelllosaurusOnto (getChebiLite), 16
getChebiLite, 16
getCellOnto (getChebiLite), 16
getCellTypeOnto (getChebiLite), 16
getDiseaseOnto (getChebiLite), 16
getGeneOnto (getChebiLite), 16
getHCAOnto (getChebiLite), 16
getLeavesFromTerm, 17
getMondoOnto (getChebiLite), 16
getOncotreeOnto (getChebiLite), 16
getOnto, 18
getPATOnto (getChebiLite), 16
getPROnto (getChebiLite), 16
getSIOOnto (getChebiLite), 16
getUBERON_NE (getChebiLite), 16

graph, 15

humrna, 19

improveNodes, 20

label_TAG (siblings_TAG), 36
labels_owlents, 20
ldfToTerms, 21
liberalMap, 22
List, 15

make_graphNEL_from_ontology_plot, 24
makeSelectInput, 23
map2prose, 24
mapOneNaive, 25, 35
minicorpus, 26

nomenCheckup, 26

onto_plot2, 27
onto_roots, 28
owl2cache, 28

packDesc2019, 29
packDesc2021, 30
packDesc2022, 30
packDesc2023, 31
parents, 31
plot.owlents, 32
print.owlents, 33
PROSYM, 33

recognizedPredicates, 34
secLevGen, 34
selectFromMap, 35
selectInput, 23
setup_entities, 35
seur3kTab, 36
show(TermSet-class), 40
show,TermSet-method(TermSet-class), 40
siblings_TAG, 36
stopWords, 37
subclasses, 38
subset_descendants, 39
sym2CellOnto, 39

TermSet-class, 40
tolower, 13

url_ok, 41
valid_ontonames, 41