Package 'ontoProc'

April 28, 2025

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 2.2.0

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

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

Depends R (>= 4.0), ontologyIndex

License Artistic-2.0

LazyLoad yes

biocViews Infrastructure, GO

RoxygenNote 7.3.2

VignetteBuilder knitr

Encoding UTF-8

Collate 'CLextend.R' 'CLfeats.R' 'available_ontos.R' 'basilisk.R' 'bind_formal_tags.R' 'bioregistry.R' 'clfixer.R' 'common_classes.R' 'connect_classes.R' 'countClasses.R' 'ctmarks.R' 'data.R' 'dropStop.R' 'fastGrep.R' 'findCommonAncestors.R' 'getOntos.R' 'get_ordo_owl_path.R' 'graphNEL.R' 'mapNaive.R' 'ontoDiff.R' 'owl2cache.R' 'owl_ops.R' 'plot.owlents.R' 'quickOnto.R' 'roots.R' 'setup_entities2.R' 'seurTab.R' 'shiny.R' 'subset_descendants.R' 'sym2CellOnto.R' 'termProc.R' 'treeproc.R'

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_21

git_last_commit a0e5d2d

git_last_commit_date 2025-04-15

Repository Bioconductor 3.21

Date/Publication 2025-04-27

Author Vincent Carey [ctb, cre] (ORCID:
<https://orcid.org/0000-0003-4046-0063>),
Sara Stankiewicz [ctb],
Victor Tarca [ctb] (ORCID: <https://orcid.org/0009-0003-1976-0392>)

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

Contents

allGOterms . . . 3
ancestors . . . 4
ancestors_names . . . 5
available_ontos . . . 5
bind_formal_tags . . . 6
bioregistry_ols_resources . . . 7
browse_ontos . . . 7
c,TermSet-method . . . 8
cellTypeToGO . . . 8
children_names . . . 9
cleanCLOnames . . . 10
CLfeats . . . 10
common_classes . . . 11
connect_classes . . . 12
ctmarks . . . 12
cyclicSigset . . . 13
demoApp . . . 14
dropStop . . . 15
fastGrep . . . 15
findCommonAncestors . . . 16
getChebiLite . . . 17
getLeavesFromTerm . . . 18
getOnto . . . 19
get_classes . . . 20
get_ordo_owl_path . . . 20
humrna . . . 21
improveNodes . . . 21
labels.owlents . . . 22

ldfToTerms . . . 22
liberalMap . . . 24
makeSelectInput . . . 25
make_graphNEL_from_ontology_plot . . . 26
map2prose . . . 26
mapOneNaive . . . 27
minicorpus . . . 28
nomenCheckup . . . 28
ontoDiff . . . 29
onto_plot2 . . . 30
onto_roots . . . 31
owl2cache . . . 31
packDesc2019 . . . 32
packDesc2021 . . . 32
packDesc2022 . . . 33
packDesc2023 . . . 33
parents . . . 34
plot.owlents . . . 35
print.owlents . . . 35
PROSYM . . . 36
quickOnto . . . 36
recognizedPredicates . . . 37
search_labels . . . 37
secLevGen . . . 38
selectFromMap . . . 39
setup_entities . . . 39
setup_entities2 . . . 40
seur3kTab . . . 41
siblings_TAG . . . 41
stopWords . . . 42
subclasses . . . 43
subset_descendants . . . 43
sym2CellOnto . . . 44
TermSet-class . . . 45
url_ok . . . 45
valid_ontonames . . . 46
[.owlents . . . 46

Index **47**

`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)
```

---

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()
o2 = try(reticulate::import("owlready2"), silent=TRUE)
if (!inherits(o2, "try-error")) {
 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()
o2 = try(reticulate::import("owlready2"), silent=TRUE)
if (!inherits(o2, "try-error")) {
 orde = setup_entities(pa)
 al = ancestors(orde[1001:1002])
 ancestors_names(al)
}
```

`available_ontos` *interrogate the cache for owl files and serialized ontologyIndex instances*

Description

interrogate the cache for owl files and serialized ontologyIndex instances

Usage

```
available_ontos(ca = BiocFileCache::BiocFileCache(), token = "owl|OIRDS")
```

Arguments

ca	defaults to BiocFileCache::BiocFileCache()
token	character(1) defaults to "owl|OIRDS"

Value

NULL if nothing relevant is found in cache, otherwise a data.frame with all cached information

---

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.

bioregistry_ols_resources *produce bioregistry_ols table*

Description

produce bioregistry_ols table

Usage

```
bioregistry_ols_resources()
```

Value

data.frame

Note

This uses the ‘resources‘ method of the bioregistry module from pip to isolate resources with a non-null ‘ols‘ component.

Examples

```
tab = bioregistry_ols_resources()
head(tab[,1:3])
```

browse_ontos *browse available ontologies with datatable*

Description

browse available ontologies with datatable

Usage

```
browse_ontos(ca = BiocFileCache::BiocFileCache(), token = "owl|OIRDS")
```

Arguments

ca	defaults to BiocFileCache::BiocFileCache()
token	character(1) defaults to "owl|OIRDS"

c,TermSet-method *combine TermSet instances*

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

```
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

```
pa = get_ordo_owl_path()
o2 = try(reticulate::import("owlready2"), silent=TRUE)
if (!inherits(o2, "try-error")) {
 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()

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: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

```
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:0001054", 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

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

Description

app to review molecular properties of cell types via cell ontology

Usage

```
ctmarks(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

```
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
   ctmarks(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

```
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 *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

```
  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

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

---

findCommonAncestors *Find common ancestors*

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()

getUBERON_NE()

getChebiOnto()

getOncotreeOnto()

getDiseaseOnto()
```

```
getGeneOnto()

getHCAOnto()

getPROnto()

getPATOnto()

getMondoOnto()

getSIOOnto()
```

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/ta
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''.

getPATOnto loads ontoRda/patoOnto.rda, produced from https://raw.githubusercontent.com/pato-ontology/pato/master/pato.obo from OBO foundry, 02-08-2019.

---

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

```
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

```
co = 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(...))'. Note: This function was revised Jul 12 2024 to allow terms that lack labels (like CHEBI references in cl.owl) to be processed, returning NA. The previous functionality which failed is available, not exported, as labelsOLD.owlents.

Examples

```
clont_path = owl2cache(url="http://purl.obolibrary.org/obo/cl.owl")
o2 = try(reticulate::import("owlready2"), silent=TRUE)
if (!inherits(o2, "try-error")) {
 clont = setup_entities(clont_path)
 labels(clont[1:5])
 labels(clont[51:55])
}
```

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)
)
```

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

```
# 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
`...`	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

```
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

```
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")
cl3k = c("CL:0000492", "CL:0001054", "CL:0000236", "CL:0000625",
   "CL:0000576", "CL:0000623", "CL:0000451", "CL:0000556")
p3k = ontologyPlot::onto_plot(cl, cl3k)
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)
```

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

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

```
#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", ...)
```

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")
```

---

ontoDiff *Display Version Differences*

Description

Highlights in green the terms that are present in the new ontology but not the old one

Usage

```
ontoDiff(newonto, oldonto, terms2use, cex = 0.8, ...)
```

Arguments

newonto	the newest version of the ontology
oldonto	the old version of the ontology
terms2use	terms of interest
cex	numeric(1) defaults to .8, supplied to Rgraphviz::graph.par
...	passed to onto_plot of ontologyPlot

Value

onto_plot2 style plot with version differences highlighted

Note

Credit to ontoPlot for the use of some of its functions.

Examples

```
cl = getOnto("diseaseOnto")
cl2 = getOnto(ontoname = "diseaseOnto", year_added = "2021")
cl3k = c("DOID:0040064","DOID:0040076","DOID:0081127","DOID:0081126","DOID:0081131","DOID:0060034")
ontoDiff(cl,cl2,cl3k)
```

---

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

```
cl = getOnto("cellOnto")
cl3k = c("CL:0000492", "CL:0001054", "CL:0000236", "CL:0000625",
   "CL:0000576", "CL:0000623", "CL:0000451", "CL:0000556")
onto_plot2(cl, cl3k)
```

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
```

owl2cache *cache an owl file accessible via URL*

Description

cache an owl file accessible via URL

Usage

```
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

```
ca = BiocFileCache::BiocFileCache()
o2 = try(reticulate::import("owlready2"), silent=TRUE)
if (!inherits(o2, "try-error")) {
 hppa = owl2cache(ca,
    url="http://purl.obolibrary.org/obo/hp/releases/2023-10-09/hp-base.owl")
 setup_entities(hppa)
}
```

`packDesc2019` *packDesc2019: overview of ontoProc resources*

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` *packDesc2021: overview of ontoProc resources*

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 *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

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

---

packDesc2023 *packDesc2023: overview of ontoProc resources*

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

```
pa = get_ordo_owl_path()
o2 = try(reticulate::import("owlready2"), silent=TRUE)
if (!inherits(o2, "try-error")) {
 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

```
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

```
cl3k = c("CL:0000492", "CL:0001054", "CL:0000236",
  "CL:0000625", "CL:0000576",
  "CL:0000623", "CL:0000451", "CL:0000556")
cl3k = gsub(":", "_", cl3k)
clont_path = owl2cache(url="http://purl.obolibrary.org/obo/cl.owl")
o2 = try(reticulate::import("owlready2"), silent=TRUE)
if (!inherits(o2, "try-error")) {
 clont = setup_entities(clont_path)
 plot(clont,cl3k)
}
```

print.owlents *short printer*

Description

short printer

Usage

```
## S3 method for class 'owlents'
print(x, ...)
```

Arguments

x	owlents instance
...	not used

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)
```

quickOnto *Probe the cache for entries found by bfcquery using the given query.*

Description

Probe the cache for entries found by bfcquery using the given query.

Usage

```
quickOnto(query, cache = BiocFileCache::BiocFileCache(), qans.only = FALSE)
```

Arguments

query	character(1), will be used by BiocFileCache::bfcquery
cache	instance of class(BiocFileCache::BiocFileCache())
qans.only	logical, defaulting to FALSE, if TRUE, just return the bfcquery result.

Note

Take only the last entry found if there are multiple hits. Use 'setup_entities2' to transform owl to ontologyIndex and cache the result if this is absent for the owl identified by query.

Examples

```
aeo = quickOnto("aeo.owl")
str(aeo)
```

---

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())
```

---

search_labels *use owlready2 ontology search facility on term labels*

Description

use owlready2 ontology search facility on term labels

Usage

```
search_labels(ontopath, regexp, case_sensitive = TRUE)
```

Arguments

ontopath	character(1) path to owl file
regexp	character(1) simple regular expression
case_sensitive	logical(1) should case be respected in search?

Value

A named list: term labels are elements, tags are names of elements. Will return NULL if nothing is found.

Examples

```
pa = get_ordo_owl_path()
ol = search_labels(pa, "*Immunog*")
orde = setup_entities2(pa)
onto_plot2(orde, names(ol))
```

`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

```
pa = get_ordo_owl_path()
o2 = try(reticulate::import("owlready2"), silent=TRUE)
if (!inherits(o2, "try-error")) {
 orde = setup_entities(pa)
 orde
 ancestors(orde[1000:1001])
 labels(orde[1000:1001])
}
```

`setup_entities2` *preparing for a small number of entry points to owlready2 mediated by basilisk, this setup function will ingest OWL, enumerate classes and their names, and produce the 'parents' list, which can then be used with ontology_index to produce a functional ontology representation*

Description

preparing for a small number of entry points to owlready2 mediated by basilisk, this setup function will ingest OWL, enumerate classes and their names, and produce the 'parents' list, which can then be used with ontology_index to produce a functional ontology representation

Usage

```
setup_entities2(owlfn, cache_object = TRUE)
```

Arguments

`owlfn`	character(1) path to OWL file
`cache_object`	logical(1) if TRUE, cache the 'ontology_index' instance in BiocFileCache::BiocFileCache()

Note

Production of an 'ontology_index' instance will often throw a warning when "Thing" is part of the ontology. suppressWarnings has been used in the code to suppress this. This may be too aggressive an approach.

Examples

```
pa = get_ordo_owl_path()
orde = setup_entities2(pa)
orde
```

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

```
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()
o2 = try(reticulate::import("owlready2"), silent=TRUE)
if (!inherits(o2, "try-error")) {
 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 *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

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

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

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

```
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

```
## 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
```

url_ok *check that a URL can get a 200 for a HEAD request*

Description

check that a URL can get a 200 for a HEAD request

Usage

```
url_ok(url)
```

Arguments

url	character(1)

Value

logical(1)

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())
```

[.owlents *subset method*

Description

subset method

Usage

```
## 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, 21
minicorpus, 28
packDesc2019, 32
packDesc2021, 32
packDesc2022, 33
packDesc2023, 33
PROSYM, 36
stopWords, 42
[.owlents, 46

adist, *29*
agrep, *9*
allGOterms, 3
ancestors, 4
ancestors_names, 5
available_ontos, 5

bind_formal_tags, 6
bioregistry_ols_resources, 7
browse_ontos, 7

c,TermSet-method, 8
cellTypeToGenes *(*cellTypeToGO*)*, 8
cellTypeToGO, 8
children_names, 9
children_TAG *(*siblings_TAG*)*, 41
cleanCLOnames, 10
CLfeats, 10
common_classes, 11
connect_classes, 12
ctmarks, 12
cyclicSigset, 13

DataFrame, *17*
demoApp, 14
dropStop, 15

fastGrep, 15
findCommonAncestors, 16

get_classes, 20
get_ordo_owl_path, 20
getCellosaurusOnto *(*getChebiLite*)*, 17
getChebiLite, 17
getChebiOnto *(*getChebiLite*)*, 17
getDiseaseOnto *(*getChebiLite*)*, 17
getGeneOnto *(*getChebiLite*)*, 17
getHCAOnto *(*getChebiLite*)*, 17
getLeavesFromTerm, 18
getMondoOnto *(*getChebiLite*)*, 17
getOncotreeOnto *(*getChebiLite*)*, 17
getOnto, 19
getPATOnto *(*getChebiLite*)*, 17
getPROnto *(*getChebiLite*)*, 17
getSIOOnto *(*getChebiLite*)*, 17
getUBERON_NE *(*getChebiLite*)*, 17
graph, *16*

humrna, 21

improveNodes, 21

label_TAG *(*siblings_TAG*)*, 41
labels.owlents, 22
ldfToTerms, 22
liberalMap, 24
List, *17*

make_graphNEL_from_ontology_plot, 26
makeSelectInput, 25
map2prose, 26
mapOneNaive, 27, *39*
minicorpus, 28

nomenCheckup, 28

onto_plot2, 30
onto_roots, 31
ontoDiff, 29
owl2cache, 31

packDesc2019, 32
packDesc2021, 32
packDesc2022, 33
packDesc2023, 33
parents, 34
plot.owlents, 35
print.owlents, 35
PROSYM, 36

quickOnto, 36

recognizedPredicates, 37

search_labels, 37
secLevGen, 38
selectFromMap, 39
selectInput, *25*
setup_entities, 39
setup_entities2, 40
seur3kTab, 41
show *(*TermSet-class*)*, 45
show,TermSet-method *(*TermSet-class*)*, 45
siblings_TAG, 41
stopWords, 42
subclasses, 43
subset_descendants, 43
sym2CellOnto, 44

TermSet-class, 45
tolower, *15*

url_ok, 45

valid_ontonames, 46