

# Package ‘ontoProc’

January 3, 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** 2.0.0

**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** basilisk.R bind\_formal\_tags.R common\_classes.R data.R getOntos.R seurTab.R CLextend.R connect\_classes.R dropStop.R graphNEL.R shiny.R treeproc.R CLfeats.R countClasses.R fastGrep.R mapNaive.R subset\_descendants.R clfixer.R ctmarks.R findCommonAncestors.R roots.R sym2CellOnto.R termProc.R owl\_ops.R get\_ordo\_owl\_path.R owl2cache.R plot.owlents.R setup\_entities2.R zzz.R bioregistry.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\_20

**git\_last\_commit** 34135d3

**git\_last\_commit\_date** 2024-10-29

**Repository** Bioconductor 3.20

**Date/Publication** 2025-01-02

**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 . . . . .	4
bind_formal_tags . . . . .	5
bioregistry_ols_resources . . . . .	6
c,TermSet-method . . . . .	6
cellTypeToGO . . . . .	7
children_names . . . . .	8
cleanCLOnames . . . . .	8
CLfeats . . . . .	9
common_classes . . . . .	9
connect_classes . . . . .	10
ctmarks . . . . .	11
cyclicSigset . . . . .	11
demoApp . . . . .	12
dropStop . . . . .	13
fastGrep . . . . .	13
findCommonAncestors . . . . .	14
getChebiLite . . . . .	15
getLeavesFromTerm . . . . .	16
getOnto . . . . .	17
get_classes . . . . .	17
get_ordo_owl_path . . . . .	18
humrna . . . . .	18
improveNodes . . . . .	19
labels.owlents . . . . .	19
ldfToTerms . . . . .	20
liberalMap . . . . .	21
makeSelectInput . . . . .	22
make_graphNEL_from_ontology_plot . . . . .	22
map2prose . . . . .	23
mapOneNaive . . . . .	24
minicorpus . . . . .	24
nomenCheckup . . . . .	25
onto_plot2 . . . . .	26
onto_roots . . . . .	26
owl2cache . . . . .	27
packDesc2019 . . . . .	27
packDesc2021 . . . . .	28
packDesc2022 . . . . .	28
packDesc2023 . . . . .	29

parents . . . . .	29
plot.owlents . . . . .	30
print.owlents . . . . .	31
PROSYM . . . . .	31
recognizedPredicates . . . . .	32
search_labels . . . . .	32
secLevGen . . . . .	33
selectFromMap . . . . .	33
setup_entities . . . . .	34
setup_entities2 . . . . .	34
seur3kTab . . . . .	35
siblings_TAG . . . . .	35
stopWords . . . . .	36
subclasses . . . . .	37
subset_descendants . . . . .	37
sym2CellOnto . . . . .	38
TermSet-class . . . . .	39
url_ok . . . . .	39
valid_ontonyms . . . . .	40
[.owlents . . . . .	40

**Index** **41**

---

allGOTerms	<i>allGOTerms: data.frame with ids and terms</i>
------------	--

---

**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)
}

```

---

bind_formal_tags	<i>add mapping from informal to formal cell type tags to a Summarized-Experiment colData</i>
------------------	--

---

**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])
```

---

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	<i>utilities for approximate matching of cell type terms to GO categories and annotations</i>
--------------	---

---

**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) <a href="#">agrep</a>
...	additional arguments to <a href="#">agrep</a>
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	<i>obtain list of names of a set of subclasses/children</i>
----------------	---

---

**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)
}
```

---

cleanCLNames	<i>obtain named character vector of terms from Cell Line Ontology, omitting obsolete and trailing ‘cell’</i>
--------------	--

---

**Description**

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

**Usage**

```
cleanCLNames()
```

**Value**

character()

**Examples**

```
cleanCLNames()[1:10]
```



---

CLfeats	<i>produce a data.frame of features relevant to a Cell Ontology class</i>
---------	---

---

**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**

```
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	<i>list and count samples with common ontological annotation in two SEs</i>
----------------	---

---

**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 'cname', 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	<i>connect ontological categories between related, annotated SummarizedExperiments</i>
-----------------	--

---

**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	<i>app to review molecular properties of cell types via cell ontology</i>
---------	---

---

### 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	<i>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</i>
--------------	--

---

### 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	<i>dropStop is a utility for removing certain words from text data</i>
----------	--

---

**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 <a href="#">tolower</a>
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	<i>some fields of interest are lists, and grep per se should not be used – this function checks and uses grep within vapply when appropriate</i>
----------	--

---

**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 <a href="#">graph</a> 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,"descendants"][[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()

getSIOnto()
```

**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/2019-02-11>, 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.

---

<code>getLeavesFromTerm</code>	<i>obtain childless descendents of a term (including query)</i>
--------------------------------	---

---

**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	<i>get the ontology based on a short tag and year</i>
---------	---

---

**Description**

get the ontology based on a short tag and year

**Usage**

```
getOnto(ontname = "cellOnto", year_added = "2023")
```

**Arguments**

ontname	character(1) must be an element in 'valid_ontnames()'
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	<i>return a generator with ontology classes</i>
-------------	---

---

**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	<i>inject linefeeds for node names for graph, with textual annotation from ontology</i>
--------------	---

---

**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	<i>retrieve labels with names</i>
----------------	-----------------------------------

---

**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	<i>use output of cyclicSigset to generate a series of character vectors constituting OBO terms</i>
------------	--

---

### 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
    "is_a: CL:0000099 ! interneuron",
    "intersection_of: CL:0000099 ! interneuron")
)
}
tms = ldfToTerms(ldf, pmap, sigels, makeIntnProlog)
cat(tms[[1]], sep="\n")
```

---

liberalMap	<i>Produce a data.frame with a set of naive terms mapped to all matching ontology ids and their formal terms</i>
------------	--

---

## 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 <a href="#">selectInput</a>
...	additional parameters passed to <a href="#">selectInput</a>

### 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 class `graphNEL`

**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	<i>use grep or agrep to find a match for a naive token into ontology</i>
-------------	--

---

**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	<i>minicorpus: a vector of annotation strings found in 'study title' of SRA metadata.</i>
------------	---

---

**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	<i>repair nomenclature mismatches (to curated term set) in a vector of terms</i>
--------------	--

---

**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 <a href="#">adist</a>

**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, cleanCL0names(), 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("cell10nto")
c13k = c("CL:0000492", "CL:0001054", "CL:0000236", "CL:0000625",
        "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
```

---

`owl2cache`*cache an owl file accessible via URL*

---

**Description**

cache an owl file accessible via URL

**Usage**

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

**Arguments**

<code>cache</code>	BiocFileCache instance or equivalent
<code>url</code>	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	<i>packDesc2021: overview of ontoProc resources</i>
--------------	---

---

**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	<i>packDesc2022: overview of ontoProc resources</i>
--------------	---

---

**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	<i>PROSYM: HGNC symbol synonyms for PR (protein ontology) entries identified in Cell Ontology</i>
--------	---

---

**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())
```

---

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	<i>simple generation of children of 'choices' given as terms, returned as TermSet</i>
-----------	---

---

**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	<i>select a set of elements from a term 'map' and return a contribution to a data.frame</i>
---------------	---

---

**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 <a href="#">mapOneNaive</a>
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	<i>construct owlents instance from an owl file</i>
----------------	--

---

**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 cnames ( 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	<i>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</i>
-----------------	---

---

**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()
```

**Examples**

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

---

seur3kTab	<i>tabulate the basic outcome of PBMC 3K tutorial of Seurat</i>
-----------	---

---

**Description**

tabulate the basic outcome of PBMC 3K tutorial of Seurat

**Usage**

```
seur3kTab()
```

**Value**

a data.frame

**Examples**

```
seur3kTab()
```

---

siblings_TAG	<i>generate a TermSet with siblings of a given term, excluding that term by default</i>
--------------	---

---

**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	<i>stopWords: vector of stop words from xpo6.com</i>
-----------	--

---

**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	<i>retrieve subclass entities</i>
------------	-----------------------------------

---

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

---

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

---

**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	<i>manage ontological data with tags and a DataFrame instance</i>
---------------	---

---

**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	<i>check that a URL can get a 200 for a HEAD request</i>
--------	--

---

**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	<i>give a vector of valid 'names' of ontoProc ontologies</i>
-----------------	--

---

**Description**

give a vector of valid 'names' of ontoProc ontologies

**Usage**

```
valid_ontonames()
```

**Examples**

```
head(valid_ontonames())
```

---

[.owlents	<i>subset method</i>
-----------	----------------------

---

**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, [18](#)
  - minicorpus, [24](#)
  - packDesc2019, [27](#)
  - packDesc2021, [28](#)
  - packDesc2022, [28](#)
  - packDesc2023, [29](#)
  - PROSYM, [31](#)
  - stopWords, [36](#)
- [.owlents, [40](#)
- adist, [25](#)
- agrep, [7](#)
- allGOterms, [3](#)
- ancestors, [4](#)
- ancestors\_names, [4](#)
- bind\_formal\_tags, [5](#)
- bioregistry\_ols\_resources, [6](#)
- c, TermSet-method, [6](#)
- cellTypeToGenes (cellTypeToGO), [7](#)
- cellTypeToGO, [7](#)
- children\_names, [8](#)
- children\_TAG (siblings\_TAG), [35](#)
- cleanCLNames, [8](#)
- CLfeats, [9](#)
- common\_classes, [9](#)
- connect\_classes, [10](#)
- ctmarks, [11](#)
- cyclicSigset, [11](#)
- DataFrame, [14](#)
- demoApp, [12](#)
- dropStop, [13](#)
- fastGrep, [13](#)
- findCommonAncestors, [14](#)
- get\_classes, [17](#)
- get\_ordo\_owl\_path, [18](#)
- getCellosaurusOnto (getChebiLite), [15](#)
- getChebiLite, [15](#)
- getChebiOnto (getChebiLite), [15](#)
- getDiseaseOnto (getChebiLite), [15](#)
- getGeneOnto (getChebiLite), [15](#)
- getHCAOnto (getChebiLite), [15](#)
- getLeavesFromTerm, [16](#)
- getMondoOnto (getChebiLite), [15](#)
- getOncotreeOnto (getChebiLite), [15](#)
- getOnto, [17](#)
- getPATOnto (getChebiLite), [15](#)
- getPROnto (getChebiLite), [15](#)
- getSIOOnto (getChebiLite), [15](#)
- getUBERON\_NE (getChebiLite), [15](#)
- graph, [14](#)
- humrna, [18](#)
- improveNodes, [19](#)
- label\_TAG (siblings\_TAG), [35](#)
- labels.owlents, [19](#)
- ldfToTerms, [20](#)
- liberalMap, [21](#)
- List, [14](#)
- make\_graphNEL\_from\_ontology\_plot, [22](#)
- makeSelectInput, [22](#)
- map2prose, [23](#)
- mapOneNaive, [24](#), [33](#)
- minicorpus, [24](#)
- nomenCheckup, [25](#)
- onto\_plot2, [26](#)
- onto\_roots, [26](#)
- owl2cache, [27](#)
- packDesc2019, [27](#)
- packDesc2021, [28](#)
- packDesc2022, [28](#)
- packDesc2023, [29](#)
- parents, [29](#)
- plot.owlents, [30](#)
- print.owlents, [31](#)
- PROSYM, [31](#)
- recognizedPredicates, [32](#)

search\_labels, [32](#)  
secLevGen, [33](#)  
selectFromMap, [33](#)  
selectInput, [22](#)  
setup\_entities, [34](#)  
setup\_entities2, [34](#)  
seur3kTab, [35](#)  
show (TermSet-class), [39](#)  
show, TermSet-method (TermSet-class), [39](#)  
siblings\_TAG, [35](#)  
stopWords, [36](#)  
subclasses, [37](#)  
subset\_descendants, [37](#)  
sym2CellOnto, [38](#)  
  
TermSet-class, [39](#)  
tolower, [13](#)  
  
url\_ok, [39](#)  
  
valid\_ontonames, [40](#)