

# Package ‘OmaDB’

April 12, 2022

**Title** R wrapper for the OMA REST API

**Version** 2.10.0

**Author** Klara Kaleb

**Maintainer** Klara Kaleb <klara.kaleb18@ic.ac.uk>, Adrian Altenhoff <adrian.altenhoff@inf.ethz.ch>

**Description** A package for the orthology prediction data download from OMA database.

**Depends** R (>= 3.5), httr (>= 1.2.1), plyr (>= 1.8.4)

**Imports** utils, ape, Biostrings, GenomicRanges, IRanges, methods, topGO, jsonlite

**URL** <https://github.com/DessimozLab/OmaDB>

**BugReports** <https://github.com/DessimozLab/OmaDB/issues>

**Encoding** UTF-8

**License** GPL-3

**LazyData** true

**Suggests** knitr, rmarkdown, testthat

**VignetteBuilder** knitr

**biocViews** Software, ComparativeGenomics, FunctionalGenomics, Genetics, Annotation, GO, FunctionalPrediction

**RoxygenNote** 7.1.1

**git\_url** <https://git.bioconductor.org/packages/OmaDB>

**git\_branch** RELEASE\_3\_14

**git\_last\_commit** 753099f

**git\_last\_commit\_date** 2021-10-26

**Date/Publication** 2022-04-12

**R topics documented:**

OmaDB-package	2
annotateSequence	3
formatTopGO	4
getAttribute	5
getGenome	5
getGenomePairs	6
getHOG	7
getLocus	8
getObjectAttributes	8
getOMAGroup	9
getProtein	10
getTaxonomy	11
getTopGO	11
getTree	12
getVersion	13
group	13
hog	14
mapSequence	14
orthologs	15
pairs	16
protein	17
resolveURL	18
searchProtein	18
sequence_annotation	19
sequence_map	20
setAPI	20
taxonomy	21
xref	21
\$.omadb_obj	22
<b>Index</b>	<b>23</b>

---

OmaDB-package	<i>OmaDB: A package for the orthology prediction data download from OMA database.</i>
---------------	---

---

**Description**

OmaDB is a wrapper for the REST API for the Orthologous Matrix project (OMA) which is a database for the inference of orthologs among complete genomes. For more details on the OMA project, see <https://omabrowser.org/>.

## OmaDB functions

The package contains a range of functions that are used to query the database. Some of the main functions are listed below:

- getProtein()
- getHOG()
- getOMAGroup()
- getGenomePairs()
- getTaxonomy()
- mapSequence()
- annotateSequence()
- searchProtein()

In addition to these, OmaDB features a range of functions that are used to format the retrieved data into some commonly used Bioconductor objects using packages such as GenomicRanges, Biostrings, topGO and ggtree. Some of them are listed below:

- formatTopGO()
- getGRanges()

The above functions are described in more detail in the package vignette's listed below:

- Get started with OmaDB
- Exploring Hierarchical orthologous groups with OmaDB
- Exploring Taxonomic trees with OmaDB
- Sequence Analysis with OmaDB

---

annotateSequence	<i>Map GO annotation to a sequence that is not available in the OMA Browser</i>
------------------	---

---

## Description

This function obtain Gene Ontology annotation for a given sequence that does not need to exist in the OMA Browser so far. The query sequence will analysed and a fast homology detection approach based on kmers will be used to detect the closest sequences in OMA. GO annotations for these top hits will be used to annotated the query sequence.

## Usage

```
annotateSequence(query)
```

**Arguments**

query            the sequence to be annotated, it can be either a string or an AAStrng object from the Biostrings package

**Value**

a data.frame containing the the GO annotation information of the most similar protein to the query sequence

**Examples**

```
annotateSequence(query='MNDPSLLGYPNVGPQQQQQQQQHAGLLGKGTPNALQQQLHMNQLTGIPPPGLMNNSDVHTSSNNSRQLLDQLANGNANMLNMMN')
```

---

formatTopGO	<i>Format the GO annotations data</i>
-------------	---------------------------------------

---

**Description**

The function to create a list of GO annotations that is compatible with topGO from protein objects in roma

**Usage**

```
formatTopGO(geneList, format)
```

**Arguments**

geneList        the list of OmaDB protein objects or a dataframe of ontologies to be included in the analysis - this is where the GO annotations are extracted from.

format          format for the data to be returned in - either 'GO2geneID' or 'geneID2GO'

**Value**

a list containing the GO2geneID or geneID2GO information

**Examples**

```
geneList = list(getProtein(id='YEAST01'),getProtein(id='YEAST03'))
annotations = formatTopGO(geneList,format='geneID2GO')
```

---

getAttribute	<i>Get the value for the Object Attribute</i>
--------------	---

---

**Description**

The function to obtain the value for an object attribute.

**Usage**

```
getAttribute(obj, attribute)
```

**Arguments**

obj	the object of interest
attribute	the attribute of interest

**Value**

an value for a given object attribute

**Examples**

```
members = getAttribute(getOMAGroup(id = 'YEAST58'), 'members')
```

---

getGenome	<i>Retrieve a genome from the OMA Browser database</i>
-----------	--

---

**Description**

This function obtains the basic information for one specific genome available on the OMA Browser, or - if no id is provided - a dataframe with all available genomes.

**Usage**

```
getGenome(id = NULL, attribute = NULL)
```

**Arguments**

id	A genome identifier. By default, all available genomes will be returned.
attribute	An extra attribute to be returned (proteins)

**Details**

Ids can be either the scientific name of a species, the NCBI taxonomy id or the UniProtKB mnemonic species code.

The optional argument `attribute` can be used to directly load the proteins belonging to the genome. Alternatively, you can access the `proteins` attribute of the result which will transparently load the proteins from the OMA Browser.

**Value**

an object containing the JSON keys as attributes or a dataframe

**Examples**

```
getGenome()
getGenome(id='HUMAN')
getGenome(id=9606)
getGenome(id='HUMAN',attribute='proteins')
```

---

getGenomePairs	<i>Retrieves the pairwise relations among two genomes</i>
----------------	---

---

**Description**

This function retrieves the pairwise relations among two genomes from the OMA Browser database. The relations are orthologs in case the genomes are different and "close paralogs" and "homoeologs" in case they are the same.

**Usage**

```
getGenomePairs(genome_id1, genome_id2, chr1 = NULL, chr2 = NULL,
               rel_type = NULL, ...)
```

**Arguments**

genome_id1	an identifier for the first genome, which can be either its taxon id or UniProt species code
genome_id2	an an identifier for the second genome, which can be either its taxon id or UniProt species code
chr1	the chromosome of interest for the first genome
chr2	the chromosome of interest for the second genome
rel_type	the pairs relationship type
...	qwargs

**Details**

By using the parameters `chr1` and `chr2`, one can limit the relations to a certain chromosome for one or both genomes. The id of the chromosome corresponds to the chromosome ids from the [getGenome](#) result.

The `rel_type` parameter further limits the returned relations to a specific subtype of orthologs (i.e. "1:1", "1:n", "m:1", "m:n") or - within a genome to either "close paralogs" or "homeologs".

**Value**

a dataframe containing information about both the entries in the orthologous pair and their relationship

**Examples**

```
getGenomePairs(genome_id1='YEAST',genome_id2='ASHG0')
```

---

getHOG

*Retrieve a HOG from the OMA Browser*

---

**Description**

The function retrieves a specific Hierarchical Orthologous Group (HOG) from the OMA Browser database. A HOG is a set of genes that have all descended from a single ancestral gene at a specific taxonomic level.

**Usage**

```
getHOG(id, level = NULL, members = FALSE)
```

**Arguments**

<code>id</code>	an identifier for the HOG to be returned - either its HOG ID or a protein id.
<code>level</code>	a specific level for the HOG to be restricted to. <code>level</code> can either be 'root', or the name of a taxonomic level that is part of the HOG, e.g. 'Fungi'. By default it will retrieve the deepest level of the most specific subhog for the given ID.
<code>members</code>	boolean that when set to TRUE returns a dataframe containing the protein members at a given hog level

**Details**

A HOG can be identified by its member proteins and a taxonomic level, or a HOG ID. As a taxonomic level, you can use either 'root' to retrieve the HOG at its deepest level, or the name of NCBI taxonomy level, or leave it out in which case the deepest level that doesn't include a duplication node is used.

The function either returns a single hog object or a list of hog objects. The latter happens if the HOG ID you provide has already split into several sub-hogs at the level you indicate.

**Value**

an object containing HOG attributes, or a list of those

**Examples**

```
getHOG(id = 'YEAST590')
getHOG(id = 'YEAST590', level='root')
getHOG(id = 'YEAST590', level='Saccharomycetaceae', members=TRUE)
```

---

getLocus	<i>Get loci for a given list of proteins</i>
----------	--

---

**Description**

Function to obtain loci in genomic range format for a given list of proteins

**Usage**

```
getLocus(proteins)
```

**Arguments**

proteins            the dataframe or a list of dataframes containing the protein data of interest. this can either be the members df or a list of protein ids.

**Value**

genomic range object from the GenomicRanges package in Bioconductor

**Examples**

```
loci = getLocus(proteins = getOMAGroup('YEAST58')['members'])
```

---

getObjectAttributes	<i>Get the Object Attributes</i>
---------------------	----------------------------------

---

**Description**

The function to obtain the attributes and their data types for the object created.

**Usage**

```
getObjectAttributes(obj)
```

**Arguments**

obj                    the object of interest



**Value**

an list of object attributes and their data classes

**Examples**

```
attributes = getObjectAttributes(getOMAGroup(id ='YEAST58'))
```

---

getOMAGroup

*Retrieve an OMA Group from the OMA Browser*

---

**Description**

This function obtains an OMA Group from the OMA Browser database. An OMA Group is defined to be a clique of proteins that are all orthologous to each other, i.e. they are all related through speciation events only. An OMA Group can thus by definition not contain any inparalogs. It is a very stringent orthology grouping approach. OMA Groups are mostly useful to infer phylogenetic species tree where they can be used as marker genes.

**Usage**

```
getOMAGroup(id, attribute = NULL)
```

**Arguments**

**id** An identifier for the group. See above for possible types of IDs.  
**attribute** an extra attribute to be returned (close\_groups)

**Details**

Retrieving an OMA Group can be done using a group nr as id, its fingerprint (a 7mer AA sequence which is unique to proteins in that group), a member protein id or any sequence pattern that is unique to the group.

**Value**

an object containing the JSON keys as attributes or a dataframe

**Examples**

```
getOMAGroup(id='58')  
getOMAGroup(id='P12345')  
getOMAGroup(id='NNRRGRI')  
getOMAGroup(id='58', attribute='close_groups')
```

---

`getProtein`*Retrieve a protein from the OMA Browser*

---

### Description

This function enables to retrieve information on one or several proteins from the OMA Browser database.

### Usage

```
getProtein(id, attribute = NULL)
```

### Arguments

<code>id</code>	Identifier(s) for the entry or entries to be returned. a character string if single entry or a vector if multiple.
<code>attribute</code>	Instead of the protein, return the attribute property of the protein. Attribute needs to be one of 'domains', 'orthologs', 'gene_ontology', 'locus', or 'homoeologs'.

### Details

In its simplest form the function returns the base data of the query protein. The query protein can be selected with any unique id, for example with a UniProtKB accession (P12345), an OMA id (YEAST00012), or a RefSeq id (NP\_001226). To retrieve more than one protein, you should pass a vector of IDs.

Non-scalar properties of proteins such as their domains, GO annotations, orthologs or homeologs will get loaded upon accessing them, or if you only need this information you can set the attribute parameter to the property name and retrieve this information directly.

### Value

An object containing the JSON keys as attributes or a dataframe containing the non-scalar protein property.

### See Also

For non-unique non-unique IDs or partial ID lookup, use [searchProtein](#) instead.

### Examples

```
getProtein(id='YEAST00001')
getProtein(id='YEAST00001', attribute='orthologs')
getProtein(id=c('YEAST00001', 'YEAST00002', 'YEAST00012'))
getProtein(id=c('YEAST00001', 'YEAST00002', 'YEAST00012'), attribute='gene_ontology')
```

---

getTaxonomy                      *Get the Taxonomic tree function*

---

### Description

The function to obtain the taxonomic tree from the database in the newick format that can be plugged into phylo.io for visualisation.

### Usage

```
getTaxonomy(root = NULL, members = NULL, newick = TRUE)
```

### Arguments

root	optional parameter, the root of the node of interest
members	optional parameter, list of member ncbi taxon or UniProt IDs that should be included in the induced taxonomy.
newick	optional parameter, boolean default set to TRUE

### Value

an object containing the JSON keys as attributes

### Examples

```
getTaxonomy()
getTaxonomy(members='YEAST,ASHGO')
getTaxonomy(root='Alveolata')
```

---

getTopGO                              *Get the topGO Object function*

---

### Description

The function to create a topGO object containing the GO annotations for the given protein list.

### Usage

```
getTopGO(annotations, format, foregroundGenes, ontology)
```

### Arguments

annotations	list of GO annotations obtained from the formatTopGO()
format	Format for the data to be returned in - either 'GO2geneID' or 'geneID2GO'
foregroundGenes	List of identifiers for the foreground genes
ontology	The ontology for which the enrichment should be done. This parameter is passed directly to the topGOdata constructor.

**Value**

topGO object

**Examples**

```
geneList = list(getProtein(id='YEAST58'),getProtein(id='YEAST00059'))
annotations = formatTopGO(geneList,format='geneID2GO')
library(topGO)
getTopGO(annotations, foregroundGenes = list('YEAST00058'), format = 'geneID2GO', ontology = 'BP')
```

---

getTree

*Get the Tree Object*

---

**Description**

A convenience function to obtain a tree object from newick tree, essentially wraps read.tree from the ape package.

**Usage**

```
getTree(newick)
```

**Arguments**

newick            The newick tree to be instantiated.

**Value**

a tree object

**Examples**

```
taxonomy = getTaxonomy(root='Alveolata')
getTree(newick=taxonomy$newick)
```

---

getVersion	<i>Get the API and database version function</i>
------------	--

---

**Description**

The function to obtain the API and database version that the package is using.

**Usage**

```
getVersion()
```

**Value**

S3 object

**Examples**

```
getVersion()
```

---

group	<i>An example OMA group object.</i>
-------	-------------------------------------

---

**Description**

An object containing information for the OMA group number 737636.

**Usage**

```
group
```

**Format**

An S3 object with 4 variables:

**group\_nr** group number, not stable across releases

**fingerprint** fingerprint of the oma group, stable across releases

**related\_groups** url to the endpoint containing the list of oma groups that share some of the orthologs with this oma group

**members** list of protein members of this oma group ...

**Source**

<https://omabrowser.org/api/group/YEAST58/>

---

hog *An example HOG object.*

---

### Description

An object containing information for the HOG:0273533.1b.

### Usage

hog

### Format

An S3 object with 8 variables:

**hog\_id** hog identifier

**level** the taxonomic level of this hog

**levels\_url** url pointer to the hog information at a given level

**members\_url** url pointer to the list of gene members for this hog

**alternative\_members** a dataframe object containing the rest of the taxonomic levels in this hog

**roothog\_id** the root taxonomic level of this hog

**parent\_hogs** a dataframe containing information on the parent hogs to the current hogs

**children\_hogs** a dataframe containing information on the children hogs to the current hogs ...

### Source

<https://omabrowser.org/api/hog/HOG:0273533.1b/>

---

mapSequence *Map the Protein Sequence Function*

---

### Description

The function to identify a sequence.

### Usage

```
mapSequence(query, search = NULL, full_length = FALSE)
```

**Arguments**

<code>query</code>	the sequence to be searched, it can be either a string or an AAStrng object from the Biostrings package
<code>search</code>	argument to choose search strategy. Can be set to 'exact', 'approximate' or 'mixed'. Defaults to 'mixed', meaning first tries to find exact match. If no target can be found, uses approximate search strategy to identify query sequence in database.
<code>full_length</code>	a boolean indicating whether or not for exact matches, the query sequence must be matching the full target sequence. By default, a partial exact match is also reported as exact match.

**Value**

a data.frame containing the information of matches for the query sequence

**Examples**

```
mapSequence(query='MNDPSLLGYPNVGPQQQQQQQQHAGLLGKGTNPALQQQLHMNQLTGIPPPGLMNNSDVHTSSNNNSRQLLDQLANGNANMLNMMDNNN
mapSequence(search='mixed', query='NKLLQPTDFQQSHIAEASKSLVDCTKQALMEMADTLTDSKTAKKQOPTGDSTPSGTATNSAVSTPLTPKIELFANG
```

---

orthologs

*An example orthologs object.*

---

**Description**

A dataframe containing information for the orthologs of protein YEAST00058.

**Usage**

```
orthologs
```

**Format**

A dataframe object with 15 variables:

**entry\_nr** entry number of the ortholog  
**omaid** oma identifier of the ortholog  
**canonicalid** canonicalid of the ortholog  
**sequence\_md5** sequence\_md5 of the ortholog  
**oma\_group** oma\_group of the ortholog  
**oma\_hog\_id** hog id of the ortholog  
**chromosome** chromosomal location of the ortholog  
**locus.start** start locus of the ortholog  
**locus.end** end locus of the ortholog

**locus.strand** locus strand of the ortholog  
**is\_main\_isoform** true/false  
**rel\_type** relationship type of the ortholog to the gene  
**distance** ortholog distance  
**score** ortholog score ...

### Source

<https://omabrowser.org/api/protein/YEAST00058/orthologs>

---

pairs	<i>An example genome alignment object.</i>
-------	--

---

### Description

A dataframe containing information for the whole genome alignment of YEAST and ASHGO.

### Usage

pairs

### Format

A dataframe object with 12 variables for each member of the pair, as well some 3 additional variables:

**entry\_nr** entry number of the ortholog  
**omaid** oma identifier of the ortholog  
**canonicalid** canonicalid of the ortholog  
**sequence\_md5** sequence\_md5 of the ortholog  
**oma\_group** oma\_group of the ortholog  
**oma\_hog\_id** hog id of the ortholog  
**chromosome** chromosomal location of the ortholog  
**locus.start** start locus of the ortholog  
**locus.end** end locus of the ortholog  
**locus.strand** locus strand of the ortholog  
**is\_main\_isoform** true/false  
**rel\_type** relationship type of the ortholog to the gene  
**distance** ortholog distance  
**score** ortholog score ...

### Source

<https://omabrowser.org/api/pairs/YEAST/ASHGO/>



---

protein	<i>An example protein object.</i>
---------	-----------------------------------

---

### Description

An object containing information for the YEAST00058 protein.

### Usage

```
protein
```

### Format

A S3 object with 23 variables:

**entry\_nr** entry number of the protein  
**entry\_url** url pointer to the protein  
**omaid** oma identifier of the protein  
**canonicalid** canonicalid of the protein  
**sequence\_md5** sequence\_md5 of the protein  
**oma\_group** oma\_group of the protein  
**oma\_hog\_id** hog id of the protein  
**chromosome** chromosomal location of the protein  
**locus** GRanges object with the locus information for the protein  
**is\_main\_isoform** true/false  
**roothog\_id** root taxonomic level of the relevant hog  
**roothog\_id** taxonomic levels of the hog in which the protein is present  
**sequence\_length** length of the protein sequence  
**sequence** AAString of the protein sequence  
**cdna** DNASTring of the protein sequence  
**domains** url pointer to the list of protein domains  
**xref** url pointer to the list of protein cross references  
**orthologs** url pointer to the list of protein orthologs  
**homeologs** url pointer to the list of protein homeologs  
**gene\_ontology** url pointer to the list of protein GO ontologies  
**oma\_group\_url** url pointer to the protein oma group  
**oma\_hog\_members** url pointer to the protein hog members  
**alternative\_isoforms\_urls** list of url pointers to the protein isoforms ...

### Source

<https://omabrowser.org/api/protein/6633022/>

---

resolveURL	<i>Load data for a given url from the OMA Browser API.</i>
------------	--

---

**Description**

This function is usually not needed by users. In most circumstances an attribute containing a URL is automatically loaded when accessed. However, in case the data is transformed into a dataframe, this will no longer be true, in which case one can access the data behind this attribute using this function.

**Usage**

```
resolveURL(url)
```

**Arguments**

url	The url of interest
-----	---------------------

**Value**

a data.frame containing the information behind an URL

**Examples**

```
resolveURL('http://omabrowser.org/api/protein/YEAST58/gene_ontology/')
```

---

searchProtein	<i>Get the CrossReferences in the OMA database for a pattern</i>
---------------	--

---

**Description**

The function to list all the crossreferences that match a certain defined pattern.

**Usage**

```
searchProtein(pattern)
```

**Arguments**

pattern	the pattern to query the OMA database with - needs to be at least 3 characters long
---------	---

**Value**

a data.frame containing information on the cross references for a given pattern

**Examples**

```
searchProtein(pattern='MAL')
```

---

sequence_annotation	<i>An example dataframe containing GO annotations identified from a given sequence.</i>
---------------------	---

---

**Description**

An example dataframe containing GO annotations identified from a given sequence.

**Usage**

```
sequence_annotation
```

**Format**

A dataframe with 13 variables:

**Qualifier** qualifier of the annotation

**GO\_ID** GO term for the annotation

**With** GO term for the annotation

**Evidence** evidence for the annotation

**Date** date

**DB\_Object\_Type** identified object type

**DB\_Object\_Name** identified object name

**Aspect** aspect

**Assigned\_By** assignment of the annotation

**GO\_name** GO term name

**DB** database

**DB.Reference** database reference

**Synonym** synonym ...

**Source**

<https://omabrowser.org/api/function/?query=MNDPSLLGYPNVGPQQQQQQQQHAGLLGKGTPNALQQQLHMNQLTGIPPPGLMN>

---

sequence_map	<i>An example dataframe containing proteins identified from a given sequence.</i>
--------------	---

---

### Description

An example dataframe containing proteins identified from a given sequence.

### Usage

```
sequence_map
```

### Format

A dataframe with 3 variables:

**query** sequence that was queried

**identified\_by** type of identification

**targets** list of protein targets identified ...

### Source

<https://omabrowser.org/api/sequences/?query=MNDPSLLGYPNVGPQQQQQQQHHAGLLGKGTPNALQQQLHMNQLTGIPPPGLM>

---

setAPI	<i>Set the url to the OMA Browser API</i>
--------	---

---

### Description

Function to set the base url to the OMA Browser API. If no url is specified, the default OMA Browser API url is used.

### Usage

```
setAPI(url)
```

### Arguments

url                      Base url to the API

---

taxonomy	<i>An example newick format taxonomy object.</i>
----------	--

---

**Description**

An example newick format taxonomy object.

**Usage**

taxonomy

**Format**

An S3 with 2 variables:

**root\_taxon** sequence that was queried

**newick** taxonomy newick ...

**Source**

<https://omabrowser.org/api/taxonomy/Alveolata/?type=newick>

---

xref	<i>An example xref object.</i>
------	--------------------------------

---

**Description**

An example xref object.

**Usage**

xref

**Format**

A dataframe with 8 variables:

**xref** cross reference

**source** source of the cross reference

**entry\_nr** oma database entry number

**oma\_id** oma id of the cross reference

**genome.code** genome\_id of the cross reference

**genome.taxon\_id** taxon\_id of the cross reference

**genome.species** species of the cross reference

**genome.genome\_url** genome url pointer of the cross reference ...

**Source**

<https://omabrowser.org/api/xref/?search=MAL>

---

\$.omadb\_obj

*Resolve URLs automatically when accessed*

---

**Description**

The function to obtain further information from a given url.

**Usage**

```
## S3 method for class 'omadb_obj'  
x$name
```

**Arguments**

x	object
name	attribute

**Value**

API response behind the URL

# Index

- \* **datasets**
  - group, [13](#)
  - hog, [14](#)
  - orthologs, [15](#)
  - pairs, [16](#)
  - protein, [17](#)
  - sequence\_annotation, [19](#)
  - sequence\_map, [20](#)
  - taxonomy, [21](#)
  - xref, [21](#)
- \$.omadb\_obj, [22](#)
- annotateSequence, [3](#)
- formatTopGO, [4](#)
- getAttribute, [5](#)
- getGenome, [5](#), [7](#)
- getGenomePairs, [6](#)
- getHOG, [7](#)
- getLocus, [8](#)
- getObjectAttributes, [8](#)
- getOMAGroup, [9](#)
- getProtein, [10](#)
- getTaxonomy, [11](#)
- getTopGO, [11](#)
- getTree, [12](#)
- getVersion, [13](#)
- group, [13](#)
- hog, [14](#)
- mapSequence, [14](#)
- OmaDB (OmaDB-package), [2](#)
- OmaDB-package, [2](#)
- orthologs, [15](#)
- pairs, [16](#)
- protein, [17](#)
- resolveURL, [18](#)
- searchProtein, [10](#), [18](#)
- sequence\_annotation, [19](#)
- sequence\_map, [20](#)
- setAPI, [20](#)
- taxonomy, [21](#)
- xref, [21](#)