Package ‘OmnipathR’

March 11, 2025

Type Package

Title OmniPath web service client and more

Version 3.14.0

Description A client for the OmniPath web service (https://www.omnipathdb.org) and many other resources. It also includes functions to transform and pretty print some of the downloaded data, functions to access a number of other resources such as BioPlex, ConsensusPathDB, EVEX, Gene Ontology, Guide to Pharmacology (IUPHAR/BPS), Harmonizome, HTRIdb, Human Phenotype Ontology, InWeb InBioMap, KEGG Pathway, Pathway Commons, Ramilowski et al. 2015, RegNetwork, ReMap, TF census, TRRUST and Vinayagam et al. 2011. Furthermore, OmnipathR features a close integration with the NicheNet method for ligand activity prediction from transcriptomics data, and its R implementation `nichenetr` (available only on github).

License MIT + file LICENSE

URL `https://r.omnipathdb.org/`

BugReports `https://github.com/saezlab/OmnipathR/issues`

biocViews GraphAndNetwork, Network, Pathways, Software, ThirdPartyClient, DataImport, DataRepresentation, GeneSignaling, GeneRegulation, SystemsBiology, Transcriptomics, SingleCell, Annotation, KEGG

Encoding UTF-8

VignetteBuilder knitr

Depends R(>= 4.0)

Imports checkmate, crayon, curl, digest, dplyr(>= 1.1.0), httr, igraph, jsonlite, later, logger, lubridate, magrittr, progress, purrr, rappdirs, readr(>= 2.0.0), readxl, rlang, rmarkdown, RSQLite, R.utils, rvest, stats, stringi, stringr, tibble, tidyr, tidyselect, tools, utils, vctrs, withr, XML, xml2, yaml, zip

Suggests BiocStyle, bookdown, ggplot2, ggraph, gprofiler2, knitr, mlrMBO, parallelMap, ParamHelpers, Rgraphviz, R.matlab, sigmajs, smoof, supraHex, testthat

RoxygenNote 7.3.2

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

git_branch RELEASE_3_20

git_last_commit 3c7d7f1

git_last_commit_date 2024-10-29

Repository Bioconductor 3.20

Date/Publication 2025-03-10

Author Alberto Valdeolivas [aut] (<https://orcid.org/0000-0001-5482-9023>),
Denes Turei [cre, aut] (<https://orcid.org/0000-0002-7249-9379>),
Attila Gabor [aut] (<https://orcid.org/0000-0002-0776-1182>),
Diego Mananes [aut] (<https://orcid.org/0000-0001-7247-6794>),
Aurelien Dugourd [aut] (<https://orcid.org/0000-0002-0714-028X>)

Maintainer Denes Turei <`turei.denes@gmail.com`>

Contents

`.omnipathr_options_defaults` *Default values for the package options*

Description

These options describe the default settings for OmnipathR so you do not need to pass these parameters at each function call. Currently the only option useful for the public web service at omnipathdb.org is “omnipathr.license“. If you are a for-profit user set it to “’commercial’“ to make sure all the data you download from OmniPath is legally allowed for commercial use. Otherwise just leave it as it is: “’academic’“. If you don’t use omnipathdb.org but within your organization you deployed your own pypath server and want to share data whith a limited availability to outside users, you may want to use a password. For this you can use the “omnipathr.password“ option. Also if you want the R package to work from another pypath server instead of omnipathdb.org, you can change the option “omnipathr.url“.

Usage

```
.omnipathr_options_defaults
```

Format

An object of class `list` of length 25.

Value

Nothing, this is not a function but a list.

`all_uniprots` *A table with all UniProt records*

Description

Retrieves a table from UniProt with all proteins for a certain organism.

Usage

```
all_uniprots(fields = "accession", reviewed = TRUE, organism = 9606L)
```

Arguments

`fields`	Character vector of fields as defined by UniProt. For possible values please refer to https://www.uniprot.org/help/return_fields
`reviewed`	Retrieve only reviewed (‘TRUE‘), only unreviewed (‘FALSE‘) or both (‘NULL‘).
`organism`	Character or integer: name or identifier of the organism.

Value

Data frame (tibble) with the requested UniProt entries and fields.

Examples

```
human_swissprot_entries <- all_uniprots(fields = 'id')
human_swissprot_entries
# # A tibble: 20,396 x 1
#    `Entry name`
#    <chr>
#  1 OR4K3_HUMAN
#  2 O52A1_HUMAN
#  3 O2AG1_HUMAN
#  4 O10S1_HUMAN
#  5 O11G2_HUMAN
# # . with 20,386 more rows
```

`all_uniprot_acs` *All UniProt ACs for one organism*

Description

All UniProt ACs for one organism

Usage

```
all_uniprot_acs(organism = 9606, reviewed = TRUE)
```

Arguments

`organism`	Character or integer: name or identifier of the organism.
`reviewed`	Retrieve only reviewed (‘TRUE‘), only unreviewed (‘FALSE‘) or both (‘NULL‘).

Value

Character vector of UniProt accession numbers.

Examples

```
human_swissprot_acs <- all_uniprot_acs()
human_swissprot_acs[1:5]
# [1] ”P51451” ”A6H8Y1” ”O60885” ”Q9Y3X0” ”P22223”
length(human_swissprot_acs)
# [1] 20376
mouse_swissprot_acs <- all_uniprot_acs(”mouse”)
```

`ancestors` *All ancestors in the ontology tree*

Description

Starting from the selected nodes, recursively walks the ontology tree until it reaches the root. Collects all visited nodes, which are the ancestors (parents) of the starting nodes.

Usage

```
ancestors(
  terms,
  db_key = "go_basic",
  ids = TRUE,
 relations = c("is_a", "part_of", "occurs_in", "regulates", "positively_regulates",
    "negatively_regulates")
)
```

Arguments

`terms`	Character vector of ontology term IDs or names. A mixture of IDs and names can be provided.
`db_key`	Character: key to identify the ontology database. For the available keys see `omnipath_show_db`.
`ids`	Logical: whether to return IDs or term names.
`relations`	Character vector of ontology relation types. Only these relations will be used.

Details

Note: this function relies on the database manager, the first call might take long because of the database load process. Subsequent calls within a short period should be faster. See `get_ontology_db`.

Value

Character vector of ontology IDs. If the input terms are all root nodes, `NULL` is returned. The starting nodes won't be included in the result unless some of them are ancestors of other starting nodes.

Examples

```
ancestors('GO:0005035', ids = FALSE)
# [1] "molecular_function"
# [2] "transmembrane signaling receptor activity"
# [3] "signaling receptor activity"
# [4] "molecular transducer activity"
```

`annotated_network` *Network interactions with annotations*

Description

Annotations are often useful in a network context, e.g. one might want to label the interacting partners by their pathway membership. This function takes a network data frame and joins an annotation data frame from both the left and the right side, so both the source and target molecular entities will be labeled by their annotations. If one entity has many annotations these will yield many rows, hence the interacting pairs won't be unique across the data frame any more. Also if one entity has really many annotations the resulting data frame might be huge, we recommend to be careful with that. Finally, if you want to do the same but with intercell annotations, there is the `import_intercell_network` function.

Usage

```
annotated_network(
  network = NULL,
  annot = NULL,
  network_args = list(),
  annot_args = list(),
  ...
)
```

Arguments

`network`	Behaviour depends on type: if list, will be passed as arguments to `omnipath_interactions` to obtain a network data frame; if a data frame or tibble, it will be used as a network data frame; if a character vector, will be assumed to be a set of resource names and interactions will be queried from these resources.
`annot`	Either the name of an annotation resource (for a list of available resources call `annotation_resources`), or an annotation data frame. If the data frame contains more than one resources, only the first one will be used.
`network_args`	List: if ‘network‘ is a resource name, pass these additional arguments to `omnipath_interactions`.
`annot_args`	List: if ‘annot‘ is a resource name, pass these additional arguments to `annotations`.
`...`	Column names selected from the annotation data frame (passed to `dplyr::select`, if empty all columns will be selected.)

Value

A data frame of interactions with annotations for both interacting entities.

Examples

```
signalink_with_pathways <-
    annotated_network("SignaLink3", "SignaLink_pathway")
```

`annotations` *Protein and gene annotations from OmniPath*

Description

Protein and gene annotations about function, localization, expression, structure and other properties, from the https://omnipathdb.org/annotations endpoint of the OmniPath web service. Note: there might be also a few miRNAs annotated; a vast majority of protein complex annotations are inferred from the annotations of the members: if all members carry the same annotation the complex inherits.

Usage

```
annotations(proteins = NULL, wide = FALSE, ...)
```

Arguments

`proteins`	Vector containing the genes or proteins for whom annotations will be retrieved (UniProt IDs or HGNC Gene Symbols or miRBase IDs). It is also possible to donwload annotations for protein complexes. To do so, write ’COMPLEX:’ right before the genesymbols of the genes integrating the complex. Check the vignette for examples.
`wide`	Convert the annotation table to wide format, which corresponds more or less to the original resource. If the data comes from more than one resource a list of wide tables will be returned. See examples at `pivot_annotations`.
`...`	Arguments passed on to `omnipath_query` `organism` Character or integer: name or NCBI Taxonomy ID of the organism. OmniPath is built of human data, and the web service provides orthology translated interactions and enzyme-substrate relationships for mouse and rat. For other organisms and query types, orthology translation will be called automatically on the downloaded human data before returning the result. `resources` Character vector: name of one or more resources. Restrict the data to these resources. For a complete list of available resources, call the ‘<query_type>_resources‘ functions for the query type of interst. `genesymbols` Character or logical: TRUE or FALS or "yes" or "no". Include the ‘genesymbols‘ column in the results. OmniPath uses UniProt IDs as the primary identifiers, gene symbols are optional. `fields` Character vector: additional fields to include in the result. For a list of available fields, call ‘query_info("interactions")‘. `default_fields` Logical: if TRUE, the default fields will be included. `silent` Logical: if TRUE, no messages will be printed. By default a summary message is printed upon successful download. `logicals` Character vector: fields to be cast to logical. `format` Character: if "json", JSON will be retrieved and processed into a nested list; any other value will return data frame. `download_args` List: parameters to pass to the download function, which is ‘readr::read_tsv‘ by default, and ‘jsonlite::safe_load‘.

`add_counts` Logical: if TRUE, the number of references and number of resources for each record will be added to the result.

`license` Character: license restrictions. By default, data from resources allowing "academic" use is returned by OmniPath. If you use the data for work in a company, you can provide "commercial" or "for-profit", which will restrict the data to those records which are supported by resources that allow for-profit use.

`password` Character: password for the OmniPath web service. You can provide a special password here which enables the use of ‘license = "ignore"‘ option, completely bypassing the license filter.

`exclude` Character vector: resource or dataset names to be excluded. The data will be filtered after download to remove records of the excluded datasets and resources.

`strict_evidences` Logical: reconstruct the "sources" and "references" columns of interaction data frames based on the "evidences" column, strictly filtering them to the queried datasets and resources. Without this, the "sources" and "references" fields for each record might contain information for datasets and resources other than the queried ones, because the downloaded records are a result of a simple filtering of an already integrated data frame.

`genesymbol_resource` Character: "uniprot" (default) or "ensembl". The OmniPath web service uses the primary gene symbols as provided by UniProt. By passing "ensembl" here, the UniProt gene symbols will be replaced by the ones used in Ensembl. This translation results in a loss of a few records, and multiplication of another few records due to ambiguous translation.

`cache` Logical: use caching, load data from and save to the. The cache directory by default belongs to the user, located in the user’s default cache directory, and named "OmnipathR". Find out about it by `getOption("omnipathr.cachedir")`. Can be changed by `omnipath_set_cachedir`.

Details

Downloading the full annotations dataset is disabled by default because the size of this data is around 1GB. We recommend to retrieve the annotations for a set of proteins or only from a few resources, depending on your interest. You can always download the full database from `https://archive.omnipathdb.org/omnipath_webservice_annotations__recent.tsv` using any standard R or `readr` method.

Value

A data frame or list of data frames:

- If `wide=FALSE` (default), all the requested resources will be in a single long format data frame.
- If `wide=TRUE`: one or more data frames with columns specific to the requested resources. If more than one resources is requested a list of data frames is returned.

See Also

- `annotation_resources`
- `pivot_annotations`
- `query_info`
- `omnipath_query`
- `annotated_network`

Examples

```
annotations <- annotations(
    proteins = c("TP53", "LMNA"),
    resources = c("HPA_subcellular")
)
```

`annotation_categories` *Annotation categories and resources*

Description

A full list of annotation resources, keys and values.

Usage

```
annotation_categories()
```

Value

A data frame with resource names, annotation key labels and for each key all possible values.

Examples

```
annot_cat <- annotation_categories()
annot_cat
# # A tibble: 46,307 x 3
#    source           label    value
#    <chr>            <chr>    <chr>
#   1 connectomeDB2020 role     ligand
#   2 connectomeDB2020 role     receptor
#   3 connectomeDB2020 location ECM
#   4 connectomeDB2020 location plasma membrane
#   5 connectomeDB2020 location secreted
#   6 KEGG-PC          pathway  Alanine, aspartate and glutamate metabolism
#   7 KEGG-PC          pathway  Amino sugar and nucleotide sugar metabolism
#   8 KEGG-PC          pathway  Aminoacyl-tRNA biosynthesis
#   9 KEGG-PC          pathway  Arachidonic acid metabolism
# 10 KEGG-PC          pathway  Arginine and proline metabolism
```

`annotation_resources` *Retrieves a list of available resources in the annotations database of OmniPath*

Description

Get the names of the resources from `https://omnipathdb.org/annotations`.

Usage

```
annotation_resources(dataset = NULL, ...)
```

Arguments

`dataset`	ignored for this query type
`...`	optional additional arguments

Value

character vector with the names of the annotation resources

See Also

- `resources`
- `annotations`

Examples

```
annotation_resources()
```

`biomart_query` *Query the Ensembl BioMart web service*

Description

Query the Ensembl BioMart web service

Usage

```
biomart_query(
  attrs = NULL,
  filters = NULL,
  transcript = FALSE,
  peptide = FALSE,
  gene = FALSE,
  dataset = "hsapiens_gene_ensembl"
)
```

Arguments

`attrs`	Character vector: one or more Ensembl attribute names.
`filters`	Character vector: one or more Ensembl filter names.
`transcript`	Logical: include Ensembl transcript IDs in the result.
`peptide`	Logical: include Ensembl peptide IDs in the result.
`gene`	Logical: include Ensembl gene IDs in the result.
`dataset`	Character: An Ensembl dataset name.

Value

Data frame with the query result

Examples

```
cel_genes <- biomart_query(
    attrs = c("external_gene_name", "start_position", "end_position"),
    gene = TRUE,
    dataset = "celegans_gene_ensembl"
)
cel_genes
# # A tibble: 46,934 × 4
#   ensembl_gene_id external_gene_name start_position end_position
#   <chr>           <chr>                       <dbl>        <dbl>
# 1 WBGene00000001  aap-1                     5107843      5110183
# 2 WBGene00000002  aat-1                     9599178      9601695
# 3 WBGene00000003  aat-2                     9244402      9246360
# 4 WBGene00000004  aat-3                     2552260      2557736
# 5 WBGene00000005  aat-4                     6272529      6275721
# # . with 46,924 more rows
```

`bioplex1` *Downloads the BioPlex version 1.0 interaction dataset*

Description

This dataset contains ~24,000 interactions detected in HEK293T cells using 2,594 baits. More details at `https://bioplex.hms.harvard.edu/interactions.php`.

Usage

```
bioplex1()
```

Value

Data frame (tibble) with interactions.

See Also

- `bioplex2`
- `bioplex3`
- `bioplex_hct116_1`
- `bioplex_all`

Examples

```
bioplex_interactions <- bioplex1()
nrow(bioplex_interactions)
# [1] 23744
colnames(bioplex_interactions)
# [1] "GeneA"         "GeneB"         "UniprotA"     "UniprotB"
# [5] "SymbolA"       "SymbolB"       "p_wrong"      "p_no_interaction"
# [9] "p_interaction"
```

`bioplex2` *Downloads the BioPlex version 2.0 interaction dataset*

Description

This dataset contains ~56,000 interactions detected in HEK293T cells using 5,891 baits. More details at https://bioplex.hms.harvard.edu/interactions.php

Usage

```
bioplex2()
```

Value

Data frame (tibble) with interactions.

See Also

- `bioplex1`
- `bioplex3`
- `bioplex_hct116_1`
- `bioplex_all`

Examples

```
bioplex_interactions <- bioplex2()
nrow(bioplex_interactions)
# [1] 56553
colnames(bioplex_interactions)
# [1] "GeneA"          "GeneB"          "UniprotA"     "UniprotB"
# [5] "SymbolA"        "SymbolB"        "p_wrong"      "p_no_interaction"
# [9] "p_interaction"
```

`bioplex3` *Downloads the BioPlex version 3.0 interaction dataset*

Description

This dataset contains ~120,000 interactions detected in HEK293T cells using 10,128 baits. More details at `https://bioplex.hms.harvard.edu/interactions.php`.

Usage

```
bioplex3()
```

Value

Data frame (tibble) with interactions.

See Also

- bioplex1
- bioplex2
- bioplex_hct116_1
- bioplex_all

Examples

```
bioplex_interactions <- bioplex3()
nrow(bioplex_interactions)
# [1] 118162
colnames(bioplex_interactions)
# [1] "GeneA"         "GeneB"         "UniprotA"     "UniprotB"
# [5] "SymbolA"       "SymbolB"       "p_wrong"      "p_no_interaction"
# [9] "p_interaction"
```

`bioplex_all` *Downloads all BioPlex interaction datasets*

Description

BioPlex provides four interaction datasets: version 1.0, 2.0, 3.0 and HCT116 version 1.0. This function downloads all of them, merges them to one data frame, removes the duplicates (based on unique pairs of UniProt IDs) and separates the isoform numbers from the UniProt IDs. More details at `https://bioplex.hms.harvard.edu/interactions.php`.

Usage

```
bioplex_all(unique = TRUE)
```

Arguments

`unique`	Logical. Collapse the duplicate interactions into single rows or keep them as they are. In case of merging duplicate records the maximum p value will be choosen for each record.

Value

Data frame (tibble) with interactions.

See Also

- bioplex1
- bioplex2
- bioplex3
- bioplex_hct116_1

Examples

```
bioplex_interactions <- bioplex_all()
bioplex_interactions
# # A tibble: 195,538 x 11
#    UniprotA IsoformA UniprotB IsoformB GeneA GeneB SymbolA SymbolB
#    <chr>       <int> <chr>       <int> <dbl> <dbl> <chr>   <chr>
#  1 A0AV02          2 Q5K4L6         NA 84561 11000 SLC12A8 SLC27A3
#  2 A0AV02          2 Q8N5V2         NA 84561 25791 SLC12A8 NGEF
#  3 A0AV02          2 Q9H6S3         NA 84561 64787 SLC12A8 EPS8L2
#  4 A0AV96          2 O00425          2 54502 10643 RBM47   IGF2BP3
#  5 A0AV96          2 O00443         NA 54502  5286 RBM47   PIK3C2A
#  6 A0AV96          2 O43426         NA 54502  8867 RBM47   SYNJ1
#  7 A0AV96          2 O75127         NA 54502 26024 RBM47   PTCD1
#  8 A0AV96          2 O95208          2 54502 22905 RBM47   EPN2
#  9 A0AV96          2 O95900         NA 54502 26995 RBM47   TRUB2
# 10 A0AV96          2 P07910          2 54502  3183 RBM47   HNRNPC
# # . with 195,528 more rows, and 3 more variables: p_wrong <dbl>,
# #   p_no_interaction <dbl>, p_interaction <dbl>
```

`bioplex_hct116_1` *Downloads the BioPlex HCT116 version 1.0 interaction dataset*

Description

This dataset contains ~71,000 interactions detected in HCT116 cells using 5,522 baits. More details at `https://bioplex.hms.harvard.edu/interactions.php`.

Usage

```
bioplex_hct116_1()
```

Value

Data frame (tibble) with interactions.

See Also

- `bioplex1`
- `bioplex2`
- `bioplex3`
- `bioplex_all`

Examples

```
bioplex_interactions <- bioplex_hct116_1()
nrow(bioplex_interactions)
# [1] 70966
colnames(bioplex_interactions)
# [1] "GeneA"          "GeneB"          "UniprotA"    "UniprotB"
# [5] "SymbolA"        "SymbolB"        "p_wrong"     "p_no_interaction"
# [9] "p_interaction"
```

`bma_motif_es` *BMA motifs from a sequence of edges*

Description

These motifs can be added to a BMA canvas.

Usage

```
bma_motif_es(edge_seq, G, granularity = 2)
```

Arguments

`edge_seq`	An igraph edge sequence.
`G`	An igraph graph object.
`granularity`	Numeric: granularity value.

Value

Character: BMA motifs as a single string.

Examples

```
interactions <- omnipath(resources = "ARN")
graph <- interaction_graph(interactions)
motifs <- bma_motif_es(igraph::E(graph)[1], graph)
```

`bma_motif_vs` *Prints a BMA motif to the screen from a sequence of nodes, which can be copy/pasted into the BMA canvas*

Description

Intended to parallel print_path_vs

Usage

```
bma_motif_vs(node_seq, G)
```

Arguments

`node_seq`	An igraph node sequence.
`G`	An igraph graph object.

Value

Character: BMA motifs as a single string.

Examples

```
interactions <- omnipath(resources = "ARN")
graph <- interaction_graph(interactions)
bma_string <- bma_motif_vs(
    igraph::all_shortest_paths(
        graph,
        from = 'ULK1',
        to = 'ATG13'
    )$res,
    graph
)
```

`chalmers_gem` *Genome scale metabolic model by Wang et al. 2021*

Description

Process the GEMs from Wang et al., 2021 (`https://github.com/SysBioChalmers`) into convenient tables.

Usage

```
chalmers_gem(organism = "Human", orphans = TRUE)
```

Arguments

`organism`	Character or integer: an organism (taxon) identifier. Supported taxons are 9606 (Homo sapiens), 10090 (Mus musculus), 10116 (Rattus norvegicus), 7955 (Danio rerio), 7227 (Drosophila melanogaster) and 6239 (Caenorhabditis elegans).
`orphans`	Logical: include orphan reactions (reactions without known enzyme).

Value

List containing the following elements:

- reactions: tibble of reaction data;
- metabolites: tibble of metabolite data;
- reaction_ids: translation table of reaction identifiers;
- metabolite_ids: translation table of metabolite identifiers;
- S: Stoichiometric matrix (sparse).

References

Wang H, Robinson JL, Kocabas P, Gustafsson J, Anton M, Cholley PE, Huang S, Gobom J, Svensson T, Uhlen M, Zetterberg H, Nielsen J. Genome-scale metabolic network reconstruction of model animals as a platform for translational research. Proc Natl Acad Sci U S A. 2021 Jul 27;118(30):e2102344118. doi: doi:10.1073/pnas.2102344118.

See Also

- `chalmers_gem_network`
- `chalmers_gem_metabolites`
- `chalmers_gem_reactions`
- `chalmers_gem_raw`
- `chalmers_gem_id_mapping_table`
- `cosmos_pkn`

Examples

```
gem <- chalmers_gem()
```

`chalmers_gem_id_mapping_table` *Metabolite ID translation tables from Chalmers Sysbio*

Description

Metabolite ID translation tables from Chalmers Sysbio

Usage

```
chalmers_gem_id_mapping_table(to, from = "metabolicatlas", organism = "Human")
```

Arguments

`to`	Character: type of ID to translate to, either label used internally in this package, or a column name from "metabolites.tsv" distributed by Chalmers Sysbio. NSE is supported.
`from`	Character: type of ID to translate from, same format as "to".
`organism`	Character or integer: name or identifier of the organism. Supported taxons are 9606 (Homo sapiens), 10090 (Mus musculus), 10116 (Rattus norvegicu), 7955 (Danio rerio), 7227 (Drosophila melanogaster) and 6239 (Caenorhabditis elegans).

Value

Tibble with two columns, "From" and "To", with the corresponding ID types.

Examples

```
chalmers_gem_id_mapping_table('metabolicatlas', 'hmdb')
```

`chalmers_gem_id_type` *Metabolite identifier type label used in Chalmers Sysbio GEM*

Description

Metabolite identifier type label used in Chalmers Sysbio GEM

Usage

```
chalmers_gem_id_type(label)
```

Arguments

`label`	Character: an ID type label, as shown in the table at `translate_ids`

Value

Character: the Chalmers GEM specific ID type label, or the input unchanged if it could not be translated (still might be a valid identifier name). These labels should be column names from the "metabolites.tsv" distributed with the GEMs.

See Also

- `hmdb_id_type`
- `uniprot_id_type`
- `ensembl_id_type`
- `uploadlists_id_type`

Examples

```
chalmers_gem_id_type("metabolicatlas")
# [1] "metsNoComp"
```

`chalmers_gem_metabolites` *Metabolites from the Chalmers SysBio GEM (Wang et al., 2021)*

Description

Metabolites from the Chalmers SysBio GEM (Wang et al., 2021)

Usage

```
chalmers_gem_metabolites(organism = "Human")
```

Arguments

`organism`	Character or integer: an organism (taxon) identifier. Supported taxons are 9606 (Homo sapiens), 10090 (Mus musculus), 10116 (Rattus norvegicu), 7955 (Danio rerio), 7227 (Drosophila melanogaster) and 6239 (Caenorhabditis elegans).

Value

Data frame of metabolite identifiers.

References

Wang H, Robinson JL, Kocabas P, Gustafsson J, Anton M, Cholley PE, Huang S, Gobom J, Svensson T, Uhlen M, Zetterberg H, Nielsen J. Genome-scale metabolic network reconstruction of model animals as a platform for translational research. Proc Natl Acad Sci U S A. 2021 Jul 27;118(30):e2102344118. doi: doi:10.1073/pnas.2102344118.

See Also

- `chalmers_gem_network`
- `chalmers_gem_reactions`
- `chalmers_gem`
- `chalmers_gem_raw`
- `chalmers_gem_id_mapping_table`
- `cosmos_pkn`

Examples

```
chalmers_gem_metabolites()
```

`chalmers_gem_network` *Chalmers SysBio GEM in the form of gene-metabolite interactions*

Description

Processing GEMs from Wang et al., 2021 (`https://github.com/SysBioChalmers`) to generate PKN for COSMOS

Usage

```
chalmers_gem_network(
  organism_or_gem = "Human",
  metab_max_degree = 400L,
  protein_ids = c("uniprot", "genesymbol"),
  metabolite_ids = c("hmdb", "kegg")
)
```

Arguments

`organism_or_gem`
Character or integer or list or data frame: either an organism (taxon) identifier or a list containing the “reactions“ data frame as it is provided by `chalmers_gem`, or the reactions data frame itself. Supported taxons are 9606 (Homo sapiens), 10090 (Mus musculus), 10116 (Rattus norvegicus), 7955 (Danio rerio), 7227 (Drosophila melanogaster) and 6239 (Caenorhabditis elegans).

`metab_max_degree`	Degree cutoff used to prune metabolites with high degree assuming they are cofactors (400 by default).
`protein_ids`	Character: translate the protein identifiers to these ID types. Each ID type results two extra columns in the output, for the "a" and "b" sides of the interaction, respectively. The default ID type for proteins is Esembl Gene ID, and by default UniProt IDs and Gene Symbols are included.
`metabolite_ids`	Character: translate the protein identifiers to these ID types. Each ID type results two extra columns in the output, for the "a" and "b" sides of the interaction, respectively. The default ID type for metabolites is Metabolic Atlas ID, and HMDB IDs and KEGG IDs are included.

Value

Data frame (tibble) of gene-metabolite interactions.

References

Wang H, Robinson JL, Kocabas P, Gustafsson J, Anton M, Cholley PE, Huang S, Gobom J, Svensson T, Uhlen M, Zetterberg H, Nielsen J. Genome-scale metabolic network reconstruction of model animals as a platform for translational research. Proc Natl Acad Sci U S A. 2021 Jul 27;118(30):e2102344118. doi: doi:10.1073/pnas.2102344118.

See Also

- `chalmers_gem`
- `chalmers_gem_metabolites`
- `chalmers_gem_reactions`
- `chalmers_gem_raw`
- `chalmers_gem_id_mapping_table`
- `cosmos_pkn`

Examples

```
gem <- chalmers_gem_network()
```

`chalmers_gem_raw` *GEM matlab file from Chalmers Sysbio (Wang et al., 2021)*

Description

Downloads and imports the matlab file containing the genome scale metabolic models created by Chalmers SysBio.

Usage

```
chalmers_gem_raw(organism = "Human")
```

Arguments

`organism`	Character or integer: name or identifier of the organism. Supported taxons are 9606 (Homo sapiens), 10090 (Mus musculus), 10116 (Rattus norvegicu), 7955 (Danio rerio), 7227 (Drosophila melanogaster) and 6239 (Caenorhabditis elegans).

Details

The Matlab object is parsed into a nested list containing a number of vectors and two sparse matrices. The top level contains a single element under the name "ihuman" for human; under this key there is an array of 31 elements. These elements are labeled by the row names of the array.

Value

Matlab object containing the GEM.

References

Wang H, Robinson JL, Kocabas P, Gustafsson J, Anton M, Cholley PE, Huang S, Gobom J, Svensson T, Uhlen M, Zetterberg H, Nielsen J. Genome-scale metabolic network reconstruction of model animals as a platform for translational research. Proc Natl Acad Sci U S A. 2021 Jul 27;118(30):e2102344118. doi: doi:10.1073/pnas.2102344118.

See Also

- `chalmers_gem_network`
- `chalmers_gem_reactions`
- `chalmers_gem`
- `chalmers_gem_reactions`
- `chalmers_gem_id_mapping_table`
- `cosmos_pkn`

Examples

```
chalmers_gem_raw()
```

`chalmers_gem_reactions` *Reactions from the Chalmers SysBio GEM (Wang et al., 2021)*

Description

Reactions from the Chalmers SysBio GEM (Wang et al., 2021)

Usage

```
chalmers_gem_reactions(organism = "Human")
```

Arguments

`organism`	Character or integer: an organism (taxon) identifier. Supported taxons are 9606 (Homo sapiens), 10090 (Mus musculus), 10116 (Rattus norvegicu), 7955 (Danio rerio), 7227 (Drosophila melanogaster) and 6239 (Caenorhabditis elegans).

Value

Data frame of reaction identifiers.

References

Wang H, Robinson JL, Kocabas P, Gustafsson J, Anton M, Cholley PE, Huang S, Gobom J, Svensson T, Uhlen M, Zetterberg H, Nielsen J. Genome-scale metabolic network reconstruction of model animals as a platform for translational research. Proc Natl Acad Sci U S A. 2021 Jul 27;118(30):e2102344118. doi: doi:10.1073/pnas.2102344118.

See Also

- `chalmers_gem_network`
- `chalmers_gem_metabolites`
- `chalmers_gem`
- `chalmers_gem_raw`
- `chalmers_gem_id_mapping_table`
- `cosmos_pkn`

Examples

```
chalmers_gem_reactions()
```

`common_name` *Common (English) names of organisms*

Description

Common (English) names of organisms

Usage

```
common_name(name)
```

Arguments

`name`	Vector with any kind of organism name or identifier, can be also mixed type.

Value

Character vector with common (English) taxon names, NA if a name in the input could not be found.

See Also

- `ncbi_taxid`
- `latin_name`
- `ensembl_name`

Examples

```
common_name(c(10090, "cjacchus", "Vicugna pacos"))
# [1] "Mouse" "White-tufted-ear marmoset" "Alpaca"
```

`complexes` *Protein complexes from OmniPath*

Description

A comprehensive dataset of protein complexes from the `https://omnipathdb.org/complexes` endpoint of the OmniPath web service.

Usage

```
complexes(...)
```

Arguments

`...`	Arguments passed on to `omnipath_query`

`organism` Character or integer: name or NCBI Taxonomy ID of the organism. OmniPath is built of human data, and the web service provides orthology translated interactions and enzyme-substrate relationships for mouse and rat. For other organisms and query types, orthology translation will be called automatically on the downloaded human data before returning the result.

`resources` Character vector: name of one or more resources. Restrict the data to these resources. For a complete list of available resources, call the '<query_type>_resources' functions for the query type of interst.

`genesymbols` Character or logical: TRUE or FALS or "yes" or "no". Include the 'genesymbols' column in the results. OmniPath uses UniProt IDs as the primary identifiers, gene symbols are optional.

`fields` Character vector: additional fields to include in the result. For a list of available fields, call 'query_info("interactions")'.

`default_fields` Logical: if TRUE, the default fields will be included.

`silent` Logical: if TRUE, no messages will be printed. By default a summary message is printed upon successful download.

`logicals` Character vector: fields to be cast to logical.

`format` Character: if "json", JSON will be retrieved and processed into a nested list; any other value will return data frame.

`download_args` List: parameters to pass to the download function, which is 'readr::read_tsv' by default, and 'jsonlite::safe_load'.

`add_counts` Logical: if TRUE, the number of references and number of resources for each record will be added to the result.

`license` Character: license restrictions. By default, data from resources allowing "academic" use is returned by OmniPath. If you use the data for work in a company, you can provide "commercial" or "for-profit", which will restrict the data to those records which are supported by resources that allow for-profit use.

`password` Character: password for the OmniPath web service. You can provide a special password here which enables the use of ‘license = "ignore"‘ option, completely bypassing the license filter.

`exclude` Character vector: resource or dataset names to be excluded. The data will be filtered after download to remove records of the excluded datasets and resources.

`strict_evidences` Logical: reconstruct the "sources" and "references" columns of interaction data frames based on the "evidences" column, strictly filtering them to the queried datasets and resources. Without this, the "sources" and "references" fields for each record might contain information for datasets and resources other than the queried ones, because the downloaded records are a result of a simple filtering of an already integrated data frame.

`genesymbol_resource` Character: "uniprot" (default) or "ensembl". The OmniPath web service uses the primary gene symbols as provided by UniProt. By passing "ensembl" here, the UniProt gene symbols will be replaced by the ones used in Ensembl. This translation results in a loss of a few records, and multiplication of another few records due to ambiguous translation.

`cache` Logical: use caching, load data from and save to the. The cache directory by default belongs to the user, located in the user’s default cache directory, and named "OmnipathR". Find out about it by `getOption("omnipathr.cachedir")`. Can be changed by `omnipath_set_cachedir`.

Value

A data frame of protein complexes.

See Also

- `complex_resources`
- `query_info`
- `omnipath_query`

Examples

```
cplx <- complexes(resources = c("CORUM", "hu.MAP"))
```

`complex_genes` *Get all the molecular complexes for a given gene(s)*

Description

This function returns all the molecular complexes where an input set of genes participate. User can choose to retrieve every complex where any of the input genes participate or just retrieve these complexes where all the genes in input set participate together.

Usage

```
complex_genes(complexes = complexes(), genes, all_genes = FALSE)
```

Arguments

`complexes`	Data frame of protein complexes (obtained using `complexes`).
`genes`	Character: search complexes where these genes present.
`all_genes`	Logical: select only complexes where all of the genes present together. By default complexes where any of the genes can be found are returned.

Value

Data frame of complexes

See Also

`complexes`

Examples

```
complexes <- complexes(resources = c("CORUM", "hu.MAP"))
query_genes <- c("LMNA", "BANF1")
complexes_with_query_genes <- complex_genes(complexes, query_genes)
```

`complex_resources` *Retrieve a list of complex resources available in Omnipath*

Description

Get the names of the resources from `https://omnipathdb.org/complexes`

Usage

```
complex_resources(dataset = NULL)
```

Arguments

`dataset`	ignored for this query type

Value

character vector with the names of the databases

See Also

- `resources`
- `complexes`

Examples

```
complex_resources()
```

`consensuspathdb_download`

Retrieves the ConsensusPathDB network

Description

Compiles a table of binary interactions from ConsensusPathDB (http://cpdb.molgen.mpg.de/) and translates the UniProtKB ACs to Gene Symbols.

Usage

```
consensuspathdb_download(complex_max_size = 4, min_score = 0.9)
```

Arguments

`complex_max_size`	Numeric: do not expand complexes with a higher number of elements than this. ConsensusPathDB does not contain conventional interactions but lists of participants, which might be members of complexes. Some records include dozens of participants and expanding them to binary interactions result thousands, sometimes hundreds of thousands of interactions from one single record. At the end, this process consumes >10GB of memory and results rather unusable data, hence it is recommended to limit the complex sizes at some low number.
`min_score`	Numeric: each record in ConsensusPathDB comes with a confidence score, expressing the amount of evidences. The default value, a minimum score of 0.9 retains approx. the top 30 percent of the interactions.

Value

Data frame (tibble) with interactions.

Examples

```
## Not run:
cpdb_data <- consensuspathdb_download(
    complex_max_size = 1,
    min_score = .99
)
nrow(cpdb_data)
# [1] 252302
colnames(cpdb_data)
# [1] ”databases”  ”references” ”uniprot_a”    ”confidence”   ”record_id”
# [6] ”uniprot_b”  ”in_complex” ”genesymbol_a” ”genesymbol_b”
cpdb_data
# # A tibble: 252,302 x 9
#    databases references uniprot_a confidence record_id uniprot_b in_com
#    <chr>     <chr>      <chr>          <dbl>     <int> <chr>     <lgl>
#  1 Reactome  NA         SUMF2_HU.     1              1 SUMF1_HU. TRUE
#  2 Reactome  NA         SUMF1_HU.     1              1 SUMF2_HU. TRUE
#  3 DIP,Reac. 22210847,. STIM1_HU.     0.998          2 TRPC1_HU. TRUE
#  4 DIP,Reac. 22210847,. TRPC1_HU.     0.998          2 STIM1_HU. TRUE
# # . with 252,292 more rows, and 2 more variables: genesymbol_a <chr>,
# #   genesymbol_b <chr
```

```
## End(Not run)
```

`consensuspathdb_raw_table` *Downloads interaction data from ConsensusPathDB*

Description

Downloads interaction data from ConsensusPathDB

Usage

```
consensuspathdb_raw_table()
```

Value

Data frame (tibble) with interactions.

Examples

```
cpdb_raw <- consensuspathdb_raw_table()
```

`cookie` *Acquire a cookie if necessary*

Description

Acquire a cookie if necessary

Usage

```
cookie(
  url,
  init_url = NULL,
  post = NULL,
  payload = NULL,
  init_post = NULL,
  init_payload = NULL,
  curl_verbose = FALSE
)
```

Arguments

`url`	Character. URL to download to get the cookie.
`init_url`	Character. An initial URL to download to get the cookie, before downloading “url“ with the cookie.
`post`	List: HTTP POST parameters.
`payload`	Data to send as payload.
`init_post`	List: HTTP POST parameters for “init_url“.
`init_payload`	Data to send as payload with “init_url“.
`curl_verbose`	Logical. Perform CURL requests in verbose mode for debugging purposes.

Value

A list with cache file path, cookies and response headers.

`cosmos_pkn` *Prior knowledge network (PKN) for COSMOS*

Description

The prior knowledge network (PKN) used by COSMOS is a network of heterogenous causal interactions: it contains protein-protein, reactant-enzyme and enzyme-product interactions. It is a combination of multiple resources:

- Genome-scale metabolic model (GEM) from Chalmers Sysbio (Wang et al., 2021.)
- Network of chemical-protein interactions from STITCH (`https://stitch.embl.de/`)
- Protein-protein interactions from Omnipath (Türei et al., 2021)

This function downloads, processes and combines the resources above. With all downloads and processing the build might take 30-40 minutes. Data is cached at various levels of processing, shortening processing times. With all data downloaded and HMDB ID translation data preprocessed, the build takes 3-4 minutes; the complete PKN is also saved in the cache, if this is available, loading it takes only a few seconds.

Usage

```
cosmos_pkn(
  organism = "human",
  protein_ids = c("uniprot", "genesymbol"),
  metabolite_ids = c("hmdb", "kegg"),
  chalmers_gem_metab_max_degree = 400L,
  stitch_score = 700L,
  ...
)
```

Arguments

`organism`	Character or integer: name or NCBI Taxonomy ID of an organism. Supported organisms vary by resource: the Chalmers GEM is available only for human, mouse, rat, fish, fly and worm. OmniPath can be translated by orthology, but for non-vertebrate or less researched taxa very few orthologues are available. STITCH is available for a large number of organisms, please refer to their web page: `https://stitch.embl.de/`.
`protein_ids`	Character: translate the protein identifiers to these ID types. Each ID type results two extra columns in the output, for the "source" and "target" sides of the interaction, respectively. The default ID type for proteins depends on the resource, hence the "source" and "target" columns are heterogenous. By default UniProt IDs and Gene Symbols are included. The Gene Symbols used in the COSMOS PKN are provided by Ensembl, and do not completely agree with the ones provided by UniProt and used in OmniPath data by default.
`metabolite_ids`	Character: translate the metabolite identifiers to these ID types. Each ID type results two extra columns in the output, for the "source" and "target" sides of the interaction, respectively. The default ID type for metabolites depends on the resource, hence the "source" and "target" columns are heterogenous. By default HMDB IDs and KEGG IDs are included.
`chalmers_gem_metab_max_degree`	Numeric: remove metabolites from the Chalmers GEM network with defgrees larger than this. Useful to remove cofactors and over-promiscuous metabolites.
`stitch_score`	Include interactions from STITCH with combined confidence score larger than this.
`...`	Further parameters to `omnipath_interactions`.

Value

A data frame of binary causal interations with effect signs, resource specific attributes and translated to the desired identifiers. The “record_id“ column identifies the original records within each resource. If one “record_id“ yields multiple records in the final data frame, it is the result of one-to-many ID translation or other processing steps. Before use, it is recommended to select one pair of ID type columns (by combining the preferred ones) and perform “distinct“ by the identifier columns and sign.

References

Wang H, Robinson JL, Kocabas P, Gustafsson J, Anton M, Cholley PE, et al. Genome-scale metabolic network reconstruction of model animals as a platform for translational research. Proceedings of the National Academy of Sciences. 2021 Jul 27;118(30):e2102344118.

Türei D, Valdeolivas A, Gul L, Palacio-Escat N, Klein M, Ivanova O, et al. Integrated intra- and intercellular signaling knowledge for multicellular omics analysis. Molecular Systems Biology. 2021 Mar;17(3):e9923.

See Also

- `chalmers_gem_network`
- `stitch_network`
- `omnipath_for_cosmos`
- `omnipath-interactions`

Examples

```
## Not run:
    human_cosmos <- cosmos_pkn(organism = "human")

## End(Not run)
```

---

curated_ligand_receptor_interactions

Curated ligand-receptor interactions

Description

The OmniPath *intercell* database annotates individual proteins and complexes, and we combine these annotations with network interactions on the client side, using `import_intercell_network`. The architecture of this database is complex, aiming to cover a broad range of knowledge on various levels of details and confidence. We can use the `intercell_consensus_filter` and `filter_intercell_network` functions for automated, data driven quality filtering, in order to enrich the cell-cell communication network in higher confidence interactions. However, for many users, a simple combination of the most established, expert curated ligand-receptor resources, provided by this function, fits better their purpose.

Usage

```
curated_ligand_receptor_interactions(
  curated_resources = c("Guide2Pharma", "HPMR", "ICELLNET", "Kirouac2010", "CellTalkDB",
    "CellChatDB", "connectomeDB2020"),
  cellphonedb = TRUE,
  cellinker = TRUE,
  talklr = TRUE,
  signalink = TRUE,
  ...
)
```

Arguments

`curated_resources`	Character vector of the resource names which are considered to be expert curated. You can include any post-translational network resource here, but if you include non ligand-receptor or non curated resources, the result will not fulfill the original intention of this function.
`cellphonedb`	Logical: include the curated interactions from *CellPhoneDB* (not the whole *CellPhoneDB* but a subset of it).
`cellinker`	Logical: include the curated interactions from *Cellinker* (not the whole *Cellinker* but a subset of it).
`talklr`	Logical: include the curated interactions from *talklr* (not the whole *talklr* but a subset of it).
`signalink`	Logical: include the ligand-receptor interactions from *SignaLink*. These are all expert curated.
`...`	Passed to `import_post_translational_interactions`: further parameters for the interaction data. Should not contain ‘resources‘ argument as that would interfere with the downstream calls.

Details

Some resources are a mixture of curated and bulk imported interactions, and sometimes it's not trivial to separate these, we take care of these here. This function does not use the *intercell* database of OmniPath, but retrieves and filters a handful of network resources. The returned data frame has the layout of *interactions* (network) data frames, and the *source* and *target* partners implicitly correspond to *ligand* and *receptor.* The data frame shows all resources and references for all interactions, but each interaction is supported by at least one ligand-receptor resource which is supposed to based on expert curation in a ligand-receptor context.

Value

A data frame similar to *interactions* (network) data frames, the *source* and *target* partners being ligand and receptor, respectively.

See Also

- `import_intercell_network`
- `filter_intercell_network`
- `annotated_network`
- `import_post_translational_interactions`
- `import_ligrecextra_interactions`
- `curated_ligrec_stats`

Examples

```
lr <- curated_ligand_receptor_interactions()
lr
```

`curated_ligrec_stats` *Statistics about literature curated ligand-receptor interactions*

Description

Statistics about literature curated ligand-receptor interactions

Usage

```
curated_ligrec_stats(...)
```

Arguments

`...`	Passed to `curated_ligand_receptor_interactions`, determines the set of all curated L-R interactions which will be compared against each of the individual resources.

Details

The data frame contains the total number of interactions, the number of interactions which overlap with the set of curated interactions *(curated_overlap),* the number of interactions with literature references from the given resource *(literature)* and the number of interactions which are curated by the given resource *(curated_self).* This latter we defined according to our best knowledge, in many cases it's not possible to distinguish curated interactions). All these numbers are also presented as a percent of the total. Importantly, here we consider interactions curated only if they've been curated in a cell-cell communication context.

Value

A data frame with estimated counts of curated ligand-receptor interactions for each L-R resource.

See Also

`curated_ligand_receptor_interactions`

Examples

```
clr <- curated_ligrec_stats()
clr
```

`database_summary` *Summary of the annotations and intercell database contents*

Description

The ‘annotations_summary‘ and ‘intercell_summary‘ query types return detailed information on the contents of these databases. It includes all the available resources, fields and values in the database.

Usage

```
database_summary(query_type, return_df = FALSE)
```

Arguments

`query_type`	Character: either "annotations" or "intercell".
`return_df`	Logical: return a data frame instead of list.

Value

Summary of the database contents: the available resources, fields, and their possible values. As a nested list if format is "json", otherwise a data frame.

Examples

```
annotations_summary <- database_summary('annotations')
```

`datasets_one_column` *Create a column with dataset names listed*

Description

From logical columns for each dataset, here we create a column that is a list of character vectors, containing dataset labels.

Usage

```
datasets_one_column(data, remove_logicals = TRUE)
```

Arguments

`data`	Interactions data frame with dataset columns (i.e. queried with the option ‘fields = "datasets"‘).
`remove_logicals`	Logical: remove the per dataset logical columns.

Value

The input data frame with the new column "datasets" added.

`descendants` *All descendants in the ontology tree*

Description

Starting from the selected nodes, recursively walks the ontology tree until it reaches the leaf nodes. Collects all visited nodes, which are the descendants (children) of the starting nodes.

Usage

```
descendants(
  terms,
  db_key = "go_basic",
  ids = TRUE,
 relations = c("is_a", "part_of", "occurs_in", "regulates", "positively_regulates",
    "negatively_regulates")
)
```

Arguments

`terms`	Character vector of ontology term IDs or names. A mixture of IDs and names can be provided.
`db_key`	Character: key to identify the ontology database. For the available keys see `omnipath_show_db`.
`ids`	Logical: whether to return IDs or term names.
`relations`	Character vector of ontology relation types. Only these relations will be used.

Details

Note: this function relies on the database manager, the first call might take long because of the database load process. Subsequent calls within a short period should be faster. See `get_ontology_db`.

Value

Character vector of ontology IDs. If the input terms are all leaves `NULL` is returned. The starting nodes won't be included in the result unless some of them are descendants of other starting nodes.

Examples

```
descendants('GO:0005035', ids = FALSE)
# [1] "tumor necrosis factor-activated receptor activity"
# [2] "TRAIL receptor activity"
# [3] "TNFSF11 receptor activity"
```

`ensembl_dataset` *Ensembl dataset name from organism*

Description

Ensembl dataset name from organism

Usage

```
ensembl_dataset(organism)
```

Arguments

`organism`	Character or integer: an organism (taxon) name or identifier. If an Ensembl dataset name is provided

Value

Character: name of an ensembl dataset.

Examples

```
ensembl_dataset(10090)
# [1] "mmusculus_gene_ensembl"
```

`ensembl_id_mapping_table` *Identifier translation table from Ensembl*

Description

Identifier translation table from Ensembl

Usage

```
ensembl_id_mapping_table(to, from = "uniprot", organism = 9606)
```

Arguments

`to`	Character or symbol: target ID type. See Details for possible values.
`from`	Character or symbol: source ID type. See Details for possible values.
`organism`	Character or integer: NCBI Taxonomy ID or name of the organism (by default 9606 for human).

Details

The arguments `to` and `from` can be provided either as character or as symbol (NSE). Their possible values are either Ensembl attribute names or synonyms listed at `translate_ids`.

Value

A data frame (tibble) with columns ‘From‘ and ‘To‘.

See Also

- `translate_ids`
- `uniprot_full_id_mapping_table`
- `uniprot_id_mapping_table`
- `hmdb_id_mapping_table`
- `chalmers_gem_id_mapping_table`

Examples

```
ensp_up <- ensembl_id_mapping_table("ensp")
ensp_up
# # A tibble: 119,129 × 2
#    From   To
#    <chr>  <chr>
#  1 P03886 ENSP00000354687
#  2 P03891 ENSP00000355046
#  3 P00395 ENSP00000354499
#  4 P00403 ENSP00000354876
#  5 P03928 ENSP00000355265
# # . with 119,124 more rows
```

`ensembl_id_type` *Ensembl identifier type label*

Description

Ensembl identifier type label

Usage

```
ensembl_id_type(label)
```

Arguments

`label`	Character: an ID type label, as shown in the table at `translate_ids`

Value

Character: the Ensembl specific ID type label, or the input unchanged if it could not be translated (still might be a valid identifier name). These labels should be valid Ensembl attribute names, directly usable in Ensembl queries.

See Also

- `uniprot_id_type`
- `uploadlists_id_type`
- `chalmers_gem_id_type`
- `hmdb_id_type`

Examples

```
ensembl_id_type("uniprot")
# [1] "uniprotswissprot"
```

`ensembl_name` *Ensembl identifiers of organisms*

Description

Ensembl identifiers of organisms

Usage

```
ensembl_name(name)
```

Arguments

`name`	Vector with any kind of organism name or identifier, can be also mixed type.

Value

Character vector with Ensembl taxon names, NA if a name in the input could not be found.

See Also

- `ncbi_taxid`
- `common_name`
- `latin_name`

Examples

```
ensembl_name(c(9606, "cat", "dog"))
# [1] "hsapiens" "fcatus" "clfamiliaris"
ensembl_name(c("human", "kitten", "cow"))
# [1] "hsapiens" NA  "btaurus"
```

`ensembl_organisms` *Organism names and identifiers from Ensembl*

Description

A table with various taxon names and identifiers: English common names, latin (scientific) names, Ensembl organism IDs and NCBI taxonomy IDs.

Usage

```
ensembl_organisms()
```

Value

A data frame with the above mentioned columns.

Examples

```
ens_org <- ensembl_organisms()
ens_org
```

`ensembl_organisms_raw` *Table of Ensembl organisms*

Description

A table with various taxon IDs and metadata about related Ensembl database contents, as shown at https://www.ensembl.org/info/about/species.html. The "Taxon ID" column contains the NCBI Taxonomy identifiers.

Usage

```
ensembl_organisms_raw()
```

Value

The table described above as a data frame.

Examples

```
ens_org <- ensembl_organisms_raw()
ens_org
```

`ensembl_orthology` *Orthologous gene pairs from Ensembl*

Description

Orthologous gene pairs from Ensembl

Usage

```
ensembl_orthology(
  organism_a = 9606,
  organism_b = 10090,
  attrs_a = NULL,
  attrs_b = NULL,
  colrename = TRUE
)
```

Arguments

`organism_a`	Character or integer: organism name or identifier for the left side organism. We query the Ensembl dataset of this organism and add the orthologues of the other organism to it. Ideally this is the organism you translate from.
`organism_b`	Character or integer: organism name or identifier for the right side organism. We add orthology information of this organism to the gene records of the left side organism.
`attrs_a`	Further attributes about organism_a genes. Will be simply added to the attributes list.

`attrs_b`	Further attributes about organism_b genes (orthologues). The available attributes are: "associated_gene_name", "chromosome", "chrom_start", "chrom_end", "wga_coverage", "goc_score", "perc_id_r1", "perc_id", "subtype". Attributes included by default: "ensembl_gene", "ensembl_peptide", "canonical_transcript_protein", "orthology_confidence" and "orthology_type".
`colrename`	Logical: replace prefixes from organism_b attribute column names, so the returned table always have the same column names, no matter the organism. E.g. for mouse these columns all have the prefix "mmusculus_homolog_", which this option changes to "b_".

Details

Only the records with orthology information are returned. The order of columns is the following: defaults of organism_a, extra attributes of organism_b, defaults of organism_b, extra attributes of organism_b.

Value

A data frame of orthologous gene pairs with gene, transcript and peptide identifiers and confidence values.

Examples

```
## Not run:
sffish <- ensembl_orthology(
    organism_b = 'Siamese fighting fish',
    attrs_a = 'external_gene_name',
    attrs_b = 'associated_gene_name'
)
sffish
# # A tibble: 175,608 × 10
#    ensembl_gene_id ensembl_transcript_id ensembl_peptide. external_gene_n.
#    <chr>           <chr>                 <chr>            <chr>
#  1 ENSG00000277196 ENST00000621424       ENSP00000481127  NA
#  2 ENSG00000277196 ENST00000615165       ENSP00000482462  NA
#  3 ENSG00000278817 ENST00000613204       ENSP00000482514  NA
#  4 ENSG00000274847 ENST00000400754       ENSP00000478910  MAFIP
#  5 ENSG00000273748 ENST00000612919       ENSP00000479921  NA
# # . with 175,603 more rows, and 6 more variables:
# #   b_ensembl_peptide <chr>, b_ensembl_gene <chr>,
# #   b_orthology_type <chr>, b_orthology_confidence <dbl>,
# #   b_canonical_transcript_protein <chr>, b_associated_gene_name <chr>
#

## End(Not run)
```

`ensure_igraph` *Converts a network to igraph object unless it is already one*

Description

Converts a network to igraph object unless it is already one

Usage

```
ensure_igraph(network)
```

Arguments

`network`	Either an OmniPath interaction data frame, or an igraph graph object.

Value

An igraph graph object.

`enzsub_graph` *Enzyme-substrate graph*

Description

Transforms the a data frame with enzyme-substrate relationships (obtained by `enzyme_substrate`) to an igraph graph object.

Usage

```
enzsub_graph(enzsub)
```

Arguments

`enzsub`	Data frame created by `enzyme_substrate`

Value

An igraph directed graph object.

See Also

- `enzyme_substrate`
- `giant_component`
- `find_all_paths`

Examples

```
enzsub <- enzyme_substrate(resources = c('PhosphoSite', 'SIGNOR'))
enzsub_g <- enzsub_graph(enzsub = enzsub)
```

`enzsub_resources` *Retrieves a list of enzyme-substrate resources available in OmniPath*

Description

Get the names of the enzyme-substrate relationship resources available in `https://omnipathdb.org/enzsub`

Usage

```
enzsub_resources(dataset = NULL)
```

Arguments

`dataset`	ignored for this query type

Value

character vector with the names of the enzyme-substrate resources

See Also

- `resources`
- `enzyme_substrate`

Examples

```
enzsub_resources()
```

`enzyme_substrate` *Enzyme-substrate (PTM) relationships from OmniPath*

Description

Imports the enzyme-substrate (more exactly, enzyme-PTM) relationship database from `https://omnipathdb.org/enzsub`. These are mostly kinase-substrate relationships, with some acetylation and other types of PTMs.

Usage

```
enzyme_substrate(...)
```

Arguments

`...`	Arguments passed on to `omnipath_query` `organism` Character or integer: name or NCBI Taxonomy ID of the organism. OmniPath is built of human data, and the web service provides orthology translated interactions and enzyme-substrate relationships for mouse and rat. For other organisms and query types, orthology translation will be called automatically on the downloaded human data before returning the result. `resources` Character vector: name of one or more resources. Restrict the data to these resources. For a complete list of available resources, call the '<query_type>_resources' functions for the query type of interst. `genesymbols` Character or logical: TRUE or FALS or "yes" or "no". Include the 'genesymbols' column in the results. OmniPath uses UniProt IDs as the primary identifiers, gene symbols are optional. `fields` Character vector: additional fields to include in the result. For a list of available fields, call 'query_info("interactions")'. `default_fields` Logical: if TRUE, the default fields will be included. `silent` Logical: if TRUE, no messages will be printed. By default a summary message is printed upon successful download. `logicals` Character vector: fields to be cast to logical. `format` Character: if "json", JSON will be retrieved and processed into a nested list; any other value will return data frame. `download_args` List: parameters to pass to the download function, which is 'readr::read_tsv' by default, and 'jsonlite::safe_load'. `add_counts` Logical: if TRUE, the number of references and number of resources for each record will be added to the result. `license` Character: license restrictions. By default, data from resources allowing "academic" use is returned by OmniPath. If you use the data for work in a company, you can provide "commercial" or "for-profit", which will restrict the data to those records which are supported by resources that allow for-profit use. `password` Character: password for the OmniPath web service. You can provide a special password here which enables the use of 'license = "ignore"' option, completely bypassing the license filter. `exclude` Character vector: resource or dataset names to be excluded. The data will be filtered after download to remove records of the excluded datasets and resources. `strict_evidences` Logical: reconstruct the "sources" and "references" columns of interaction data frames based on the "evidences" column, strictly filtering them to the queried datasets and resources. Without this, the "sources" and "references" fields for each record might contain information for datasets and resources other than the queried ones, because the downloaded records are a result of a simple filtering of an already integrated data frame. `genesymbol_resource` Character: "uniprot" (default) or "ensembl". The OmniPath web service uses the primary gene symbols as provided by UniProt. By passing "ensembl" here, the UniProt gene symbols will be replaced by the ones used in Ensembl. This translation results in a loss of a few records, and multiplication of another few records due to ambiguous translation. `cache` Logical: use caching, load data from and save to the. The cache directory by default belongs to the user, located in the user's default cache directory,

and named "OmnipathR". Find out about it by `getOption("omnipathr.cachedir")`. Can be changed by `omnipath_set_cachedir`.

Value

A data frame of enzymes and their PTM substrates.

See Also

- `enzsub_resources`
- `omnipath_interactions`
- `enzsub_graph`
- `print_interactions`
- `query_info`
- `omnipath_query`

Examples

```
enzsub <- enzyme_substrate(
    resources = c("PhosphoSite", "SIGNOR"),
    organism = 9606
)
```

`evex_download` *Interactions from the EVEX database*

Description

Downloads interactions from EVEX, a versatile text mining resource (`http://evexdb.org`). Translates the Entrez Gene IDs to Gene Symbols and combines the interactions and references into a single data frame.

Usage

```
evex_download(
    min_confidence = NULL,
    remove_negatives = TRUE,
    top_confidence = NULL
)
```

Arguments

`min_confidence`	Numeric: a threshold for confidence scores. EVEX confidence scores span roughly from -3 to 3. By providing a numeric value in this range the lower confidence interactions can be removed. If NULL no filtering performed.
`remove_negatives`	Logical: remove the records with the "negation" attribute set.
`top_confidence`	Confidence cutoff as quantile (a number between 0 and 1). If NULL no filtering performed.

Value

Data frame (tibble) with interactions.

Examples

```
evex_interactions <- evex_download()
evex_interactions
# # A tibble: 368,297 x 13
#   general_event_id source_entrezge. target_entrezge. confidence negation
#               <dbl> <chr>            <chr>                 <dbl>    <dbl>
# 1               98 8651             6774                  -1.45        0
# 2              100 8431             6774                  -1.45        0
# 3              205 6261             6263                   0.370       0
# 4              435 1044             1045                  -1.09        0
# . with 368,287 more rows, and 8 more variables: speculation <dbl>,
#   coarse_type <chr>, coarse_polarity <chr>, refined_type <chr>,
#   refined_polarity <chr>, source_genesymbol <chr>,
#   target_genesymbol <chr>, references <chr>
```

`evidences` *Show evidences for an interaction*

Description

Show evidences for an interaction

Usage

```
evidences(
  partner_a,
  partner_b,
  interactions = NULL,
  directed = FALSE,
  open = TRUE,
  browser = NULL,
  max_pages = 25L
)
```

Arguments

`partner_a`	Identifier or name of one interacting partner. The order of the partners matter only if ‘directed‘ is ‘TRUE‘. For both partners, vectors of more than one identifiers can be passed.
`partner_b`	Identifier or name of the other interacting partner.
`interactions`	An interaction data frame. If not provided, all interactions will be loaded within this function, but that takes noticeable time. If a ‘list‘ is provided, it will be used as parameters for `omnipath_interactions`. This way you can define the organism, datasets or the interaction type.
`directed`	Logical: does the direction matter? If ‘TRUE‘, only a → b interactions will be shown.
`open`	Logical: open online articles in a web browser.

`browser`	Character: override the web browser executable used to open online articles.
`max_pages`	Numeric: largest number of pages to open. This is to prevent opening hundreds or thousands of pages at once.

Details

If the number of references is larger than ‘max_pages‘, the most recent ones will be opened. URLs are passed to the browser in order of decreasing publication date, though browsers do not seem to respect the order at all. In addition Firefox, if it’s not open already, tends to randomly open empty tab for the first or last URL, have no idea what to do about it.

Value

Nothing.

Examples

```
## Not run:
evidences('CALM1', 'TRPC1', list(datasets = 'omnipath'))

## End(Not run)
```

`extra_attrs` *Extra attribute names in an interaction data frame*

Description

Interaction data frames might have an ‘extra_attrs‘ column if this field has been requested in the query by passing the ‘fields = ’extra_attrs’ argument. This column contains resource specific attributes for the interactions. The names of the attributes consist of the name of the resource and the name of the attribute, separated by an underscore. This function returns the names of the extra attributes available in the provided data frame.

Usage

```
extra_attrs(data)
```

Arguments

`data`	An interaction data frame, as provided by any of the `omnipath-interactions` functions.

Value

Character: the names of the extra attributes in the data frame.

See Also

- `extra_attrs_to_cols`
- `has_extra_attrs`
- `with_extra_attrs`
- `filter_extra_attrs`
- `extra_attr_values`

Examples

```
i <- omnipath(fields = "extra_attrs")
extra_attrs(i)
```

`extra_attrs_to_cols` *New columns from extra attributes*

Description

New columns from extra attributes

Usage

```
extra_attrs_to_cols(data, ..., flatten = FALSE, keep_empty = TRUE)
```

Arguments

`data`	An interaction data frame.
`...`	The names of the extra attributes; NSE is supported. Custom column names can be provided as argument names.
`flatten`	Logical: unnest the list column even if some records have multiple values for the attributes; these will yield multiple records in the resulted data frame.
`keep_empty`	Logical: if ‘flatten‘ is ‘TRUE‘, shall we keep the records which do not have the attribute?

Value

Data frame with the new column created; the new column is list type if one interaction might have multiple values of the attribute, or character type if

See Also

- `extra_attrs`
- `has_extra_attrs`
- `with_extra_attrs`
- `filter_extra_attrs`
- `extra_attr_values`

Examples

```
i <- omnipath(fields = "extra_attrs")
extra_attrs_to_cols(i, Cellinker_type, Macrophage_type)
extra_attrs_to_cols(
    i,
    Cellinker_type,
    Macrophage_type,
    flatten = TRUE,
    keep_empty = FALSE
)
```

`extra_attr_values` *Possible values of an extra attribute*

Description

Extracts all unique values of an extra attribute occuring in this data frame.

Usage

```
extra_attr_values(data, key)
```

Arguments

`data`	An interaction data frame with *extra_attrs* column.
`key`	The name of an extra attribute.

Details

Note, at the end we unlist the result, which means it works well for attributes which are atomic vectors but gives not so useful result if the attribute values are more complex objects. As the time of writing this, no such complex extra attribute exist in OmniPath.

Value

A vector, most likely character, with the unique values of the extra attribute occuring in the data frame.

See Also

- `extra_attrs_to_cols`
- `has_extra_attrs`
- `with_extra_attrs`
- `filter_extra_attrs`
- `extra_attrs`

Examples

```
op <- omnipath(fields = "extra_attrs")
extra_attr_values(op, SIGNOR_mechanism)
```

`filter_by_resource` *Filters OmniPath data by resources*

Description

Keeps only those records which are supported by any of the resources of interest.

Usage

```
filter_by_resource(data, resources = NULL)
```

Arguments

`data`	A data frame downloaded from the OmniPath web service (interactions, enzyme-substrate or complexes).
`resources`	Character vector with resource names to keep.

Value

The data frame filtered.

Examples

```
interactions <- omnipath()
signor <- filter_by_resource(interactions, resources = "SIGNOR")
```

`filter_evidences` *Filter evidences by dataset, resource and license*

Description

Filter evidences by dataset, resource and license

Usage

```
filter_evidences(data, ..., datasets = NULL, resources = NULL, exclude = NULL)
```

Arguments

`data`	An interaction data frame with some columns containing evidences as nested lists.
`...`	The evidences columns to filter: tidyselect syntax is supported. By default the columns "evidences", "positive", "negative", "directed" and "undirected" are filtered, if present.
`datasets`	A character vector of dataset names.
`resources`	A character vector of resource names.
`exclude`	Character vector of resource names to be excluded.

Value

The input data frame with the evidences in the selected columns filtered.

See Also

- `only_from`
- `unnest_evidences`
- `from_evidences`

`filter_extra_attrs` *Filter interactions by extra attribute values*

Description

Filter interactions by extra attribute values

Usage

```
filter_extra_attrs(data, ..., na_ok = TRUE)
```

Arguments

`data`	An interaction data frame with *extra_attrs* column.
`...`	Extra attribute names and values. The contents of the extra attribute *name* for each record will be checked against the values provided. The check by default is a set intersection: if any element is common between the user provided values and the values of the extra attribute for the record, the record will be matched. Alternatively, any value can be a custom function which accepts the value of the extra attribute and returns a single logical value. Finally, if the extra attribute name starts with a dot, the result of the check will be negated.
`na_ok`	Logical: keep the records which do not have the extra attribute. Typically these are the records which are not from the resource providing the extra attribute.

Value

The input data frame with records removed according to the filtering criteria.

See Also

- `extra_attrs`
- `has_extra_attrs`
- `extra_attrs_to_cols`
- `with_extra_attrs`
- `extra_attr_values`

Examples

```
cl <- post_translational(
    resources = "Cellinker",
    fields = "extra_attrs"
)
# Only cell adhesion interactions from Cellinker
filter_extra_attrs(cl, Cellinker_type = "Cell adhesion")

op <- omnipath(fields = "extra_attrs")
# Any mechanism except phosphorylation
filter_extra_attrs(op, .SIGNOR_mechanism = "phosphorylation")
```

`filter_intercell` *Filter intercell annotations*

Description

Filters a data frame retrieved by `intercell`.

Usage

```
filter_intercell(
  data,
  categories = NULL,
  resources = NULL,
  parent = NULL,
  scope = NULL,
  aspect = NULL,
  source = NULL,
  transmitter = NULL,
  receiver = NULL,
  secreted = NULL,
  plasma_membrane_peripheral = NULL,
  plasma_membrane_transmembrane = NULL,
  proteins = NULL,
  causality = NULL,
  topology = NULL,
  ...
)
```

Arguments

`data`	An intercell annotation data frame as provided by `intercell`.
`categories`	Character: allow only these values in the `category` column.
`resources`	Character: allow records only from these resources.
`parent`	Character: filter for records with these parent categories.
`scope`	Character: filter for records with these annotation scopes. Possible values are `generic` and `specific`.

`aspect`	Character: filter for records with these annotation aspects. Possible values are `functional` and `locational`.
`source`	Character: filter for records with these annotation sources. Possible values are `composite` and `resource_specific`.
`transmitter`	Logical: if `TRUE` only transmitters, if `FALSE` only non-transmitters will be selected, if `NULL` it has no effect.
`receiver`	Logical: works the same way as `transmitters`.
`secreted`	Logical: works the same way as `transmitters`.
`plasma_membrane_peripheral`	Logical: works the same way as `transmitters`.
`plasma_membrane_transmembrane`	Logical: works the same way as `transmitters`.
`proteins`	Character: filter for annotations of these proteins. Gene symbols or UniProt IDs can be used.
`causality`	Character: filter for records with these causal roles. Possible values are `transmitter` and `receiver`. The filter applied simultaneously to the `transmitter` and `receiver` arguments, it's just a different notation for the same thing.
`topology`	Character: filter for records with these localization topologies. Possible values are `secreced`, `plasma_membrane_peripheral` and `plasma_membrane_transmembrane`; the shorter notations `sec`, `pmp` and `pmtm` can be used. Has the same effect as the logical type arguments, just uses a different notation.
`...`	Ignored.

Value

The intercell annotation data frame filtered according to the specified conditions.

See Also

- `intercell`
- `intercell_categories`
- `intercell_generic_categories`
- `intercell_summary`
- `intercell_network`

Examples

```
ic <- intercell()
ic <- filter_intercell(
    ic,
    transmitter = TRUE,
    secreted = TRUE,
    scope = "specific"
)
```

---

filter_intercell_network

Quality filter an intercell network

Description

The intercell database of OmniPath covers a very broad range of possible ways of cell to cell communication, and the pieces of information, such as localization, topology, function and interaction, are combined from many, often independent sources. This unavoidably result some weird and unexpected combinations which are false positives in the context of intercellular communication. `intercell_network` provides a shortcut (`high_confidence`) to do basic quality filtering. For custom filtering or experimentation with the parameters we offer this function.

Usage

```
filter_intercell_network(
  network,
  transmitter_topology = c("secreted", "plasma_membrane_transmembrane",
    "plasma_membrane_peripheral"),
  receiver_topology = "plasma_membrane_transmembrane",
  min_curation_effort = 2,
  min_resources = 1,
  min_references = 0,
  min_provenances = 1,
  consensus_percentile = 50,
  loc_consensus_percentile = 30,
  ligand_receptor = FALSE,
  simplify = FALSE,
  unique_pairs = FALSE,
  omnipath = TRUE,
  ligrecextra = TRUE,
  kinaseextra = FALSE,
  pathwayextra = FALSE,
  ...
)
```

Arguments

`network`	An intercell network data frame, as provided by `intercell_network`, without `simplify`.
`transmitter_topology`	Character vector: topologies allowed for the entities in transmitter role. Abbreviations allowed: "sec", "pmtm" and "pmp".
`receiver_topology`	Same as `transmitter_topology` for the entities in the receiver role.
`min_curation_effort`	Numeric: a minimum value of curation effort (resource-reference pairs) for network interactions. Use zero to disable filtering.
`min_resources`	Numeric: minimum number of resources for interactions. The value 1 means no filtering.

`min_references`	Numeric: minimum number of references for interactions. Use zero to disable filtering.
`min_provenances`	Numeric: minimum number of provenances (either resources or references) for interactions. Use zero or one to disable filtering.
`consensus_percentile`	Numeric: percentile threshold for the consensus score of generic categories in intercell annotations. The consensus score is the number of resources supporting the classification of an entity into a category based on combined information of many resources. Here you can apply a cut-off, keeping only the annotations supported by a higher number of resources than a certain percentile of each category. If `NULL` no filtering will be performed. The value is either in the 0-1 range, or will be divided by 100 if greater than 1. The percentiles will be calculated against the generic composite categories and then will be applied to their resource specific annotations and specific child categories.
`loc_consensus_percentile`	Numeric: similar to `consensus_percentile` for major localizations. For example, with a value of 50, the secreted, plasma membrane transmembrane or peripheral attributes will be `TRUE` only where at least 50 percent of the resources support these.
`ligand_receptor`	Logical. If `TRUE`, only *ligand* and *receptor* annotations will be used instead of the more generic *transmitter* and *receiver* categories.
`simplify`	Logical: keep only the most often used columns. This function combines a network data frame with two copies of the intercell annotation data frames, all of them already having quite some columns. With this option we keep only the names of the interacting pair, their intercellular communication roles, and the minimal information of the origin of both the interaction and the annotations.
`unique_pairs`	Logical: instead of having separate rows for each pair of annotations, drop the annotations and reduce the data frame to unique interacting pairs. See `unique_intercell_network` for details.
`omnipath`	Logical: shortcut to include the *omnipath* dataset in the interactions query.
`ligrecextra`	Logical: shortcut to include the *ligrecextra* dataset in the interactions query.
`kinaseextra`	Logical: shortcut to include the *kinaseextra* dataset in the interactions query.
`pathwayextra`	Logical: shortcut to include the *pathwayextra* dataset in the interactions query.
`...`	If `simplify` or `unique_pairs` is `TRUE`, additional column names can be passed here to `dplyr::select` on the final data frame. Otherwise ignored.

Value

An intercell network data frame filtered.

See Also

- `intercell_network`
- `unique_intercell_network`
- `simplify_intercell_network`
- `intercell`
- `intercell_categories`
- `intercell_generic_categories`
- `intercell_summary`

Examples

```
icn <- intercell_network()
icn_f <- filter_intercell_network(
    icn,
    consensus_percentile = 75,
    min_provenances = 3,
    simplify = TRUE
)
```

`find_all_paths` *All paths between two groups of vertices*

Description

Finds all paths up to length ‘maxlen‘ between specified groups of vertices. This function is needed only becaues igraph‘s ‘all_shortest_paths‘ finds only the shortest, not any path up to a defined length.

Usage

```
find_all_paths(
    graph,
    start,
    end,
    attr = NULL,
    mode = 'OUT',
    maxlen = 2,
    progress = TRUE
)
```

Arguments

`graph`	An igraph graph object.
`start`	Integer or character vector with the indices or names of one or more start vertices.
`end`	Integer or character vector with the indices or names of one or more end vertices.
`attr`	Character: name of the vertex attribute to identify the vertices by. Necessary if ‘start‘ and ‘end‘ are not igraph vertex ids but for example vertex names or labels.
`mode`	Character: IN, OUT or ALL. Default is OUT.
`maxlen`	Integer: maximum length of paths in steps, i.e. if maxlen = 3, then the longest path may consist of 3 edges and 4 nodes.
`progress`	Logical: show a progress bar.

Value

List of vertex paths, each path is a character or integer vector.

See Also

- `interaction_graph`
- `enzsub_graph`
- `giant_component`

Examples

```
interactions <- import_omnipath_interactions()
graph <- interaction_graph(interactions)
paths <- find_all_paths(
    graph = graph,
    start = c('EGFR', 'STAT3'),
    end = c('AKT1', 'ULK1'),
    attr = 'name'
)
```

`from_evidences` *Recreate interaction records from evidences columns*

Description

Recreate interaction records from evidences columns

Usage

```
from_evidences(data, .keep = FALSE)
```

Arguments

`data`	An interaction data frame from the OmniPath web service with evidences column.
`.keep`	Logical: keep the original "evidences" column when unnesting to separate columns by direction.

Details

The OmniPath interaction data frames specify interactions primarily by three columns: "is_directed", "is_stimulation" and "is_inhibition". Besides these, there are the "sources" and "references" columns that are always included in data frames created by OmnipathR and list the resources and literature references for each interaction, respectively. The optional "evidences" column is required to find out which of the resources and references support the direction or effect sign of the interaction. To properly recover information for arbitrary subsets of resources or datasets, the evidences can be filtered first, and then the standard data frame columns can be reconstructed from the selected evidences. This function is able to do the latter. It expects either an "evidences" column or evidences in their wide format 4 columns layout. It overwrites the standard columns of interaction records based on data extracted from the evidences, including the "curation_effort" and "consensus..." columns.

Note: The "curation_effort" might be calculated slightly differently from the version included in the OmniPath web service. Here we count the resources and the also add the number of references for each resource. E.g. a resource without any literatur reference counts as 1, while a resource with 3 references adds 4 to the value of the curation effort.

Note: If the "evidences" column has been already unnested to multiple columns ("positive", "negative", etc.) by `unnest_evidences`, then these will be used; otherwise, the column will be unnested within this function.

Note: This function (or rather its wrapper, `only_from`) is automatically applied if the ‘strict_evidences‘ argument is passed to any function querying interactions (see `omnipath-interactions`).

Value

A copy of the input data frame with all the standard columns describing the direction, effect, resources and references of the interactions recreated based on the contents of the nested list evidences column(s).

See Also

- `filter_evidences`
- `unnest_evidences`
- `only_from`

Examples

```
## Not run:
ci <- collectri(evidences = TRUE)
ci <- unnest_evidences(ci)
ci <- filter_evidences(datasets = 'collectri')
ci <- from_evidences(ci)
# the three lines above are equivalent to only_from(ci)
# and all the four lines above is equivalent to:
# collectri(strict_evidences = TRUE)

## End(Not run)
```

`get_db` *Access a built in database*

Description

Databases are resources which might be costly to load but can be used many times by functions which usually automatically load and retrieve them from the database manager. Each database has a lifetime and will be unloaded automatically upon expiry.

Usage

```
get_db(key, param = NULL, reload = FALSE, ...)
```

Arguments

`key`	Character: the key of the database to load. For a list of available keys see `omnipath_show_db`.

`param`	List: override the defaults or pass further parameters to the database loader function. See the loader functions and their default parameters in `omnipath_show_db`. If the database is already loaded with different parameters it will be reloaded with the new parameters only if the `reload` option is `TRUE`.
`reload`	Reload the database if `param` passed here is different from the parameters used the last time the database was loaded. If different functions with different parameters access the database repeatedly and request reload the frequent reloads might cost substantial time and resource use.
`...`	Arguments for the loader function of the database. These override the default arguments.

Value

An object with the database contents. The exact format depends on the database, most often it is a data frame or a list.

See Also

`omnipath_show_db`.

Examples

```
organisms <- get_db('organisms')
```

`get_ontology_db` *Access an ontology database*

Description

Retrieves an ontology database with relations in the desired data structure. The database is automatically loaded and the requested data structure is constructed if necessary. The databases stay loaded up to a certain time period (see the option `omnipathr.db_lifetime`). Hence the first one of repeated calls to this function might take long and the subsequent ones should be really quick.

Usage

```
get_ontology_db(key, rel_fmt = "tbl", child_parents = TRUE)
```

Arguments

`key`	Character: key of the ontology database. For the available keys see `omnipath_show_db`.
`rel_fmt`	Character: the data structure of the ontology relations. Posible values are 1) "tbl" a data frame, 2) "lst" a list or 3) "gra" a graph.
`child_parents`	Logical: whether the ontology relations should point from child to parents (`TRUE`) or from parent to children (`FALSE`).

Value

A list with the following elements: 1) "names" a table with term IDs and names; 2) "namespaces" a table to connect term IDs and namespaces they belong to; 3) "relations" a table with relations between terms and their parent terms; 4) "subsets" a table with terms and the subsets they are part of; 5) "obsolete" character vector with all the terms labeled as obsolete.

See Also

- omnipath_show_db
- get_db

Examples

```
go <- get_ontology_db('go_basic', child_parents = FALSE)
```

---

giant_component *Giant component of a graph*

Description

For an igraph graph object returns its giant component.

Usage

```
giant_component(graph)
```

Arguments

graph	An igraph graph object.

Value

An igraph graph object containing only the giant component.

Examples

```
interactions <- import_post_translational_interactions()
graph <- interaction_graph(interactions)
graph_gc <- giant_component(graph)
```

---

go_annot_download *Gene annotations from Gene Ontology*

Description

Gene Ontology is an ontology of gene subcellular localizations, molecular functions and involvement in biological processes. Gene products across many organisms are annotated with the ontology terms. This function downloads the gene-ontology term associations for certain model organisms or all organisms. For a description of the columns see http://geneontology.org/docs/go-annotation-file-gaf-format-2.2/.

Usage

```
go_annot_download(organism = "human", aspects = c("C", "F", "P"), slim = NULL)
```

Arguments

`organism`	Character: either "chicken", "cow", "dog", "human", "pig" or "uniprot_all".
`aspects`	Character vector with some of the following elements: "C" (cellular component), "F" (molecular function) and "P" (biological process). Gene Ontology is three separate ontologies called as three aspects. By this parameter you can control which aspects to include in the output.
`slim`	Character: if not `NULL`, the name of a GOsubset (slim). instead of the full GO annotation, the slim annotation will be returned. See details at `go_annot_slim`. If `TRUE`, the "generic" slim will be used.

Value

A tibble (data frame) of annotations as it is provided by the database

Examples

```
goa_data <- go_annot_download()
goa_data
# # A tibble: 606,840 x 17
#    db       db_object_id db_object_symbol qualifier go_id   db_ref
#    <fct>    <chr>        <chr>            <fct>     <chr>   <chr>
#   1 UniProt. A0A024RBG1   NUDT4B           NA        GO:000. GO_REF:00.
#   2 UniProt. A0A024RBG1   NUDT4B           NA        GO:000. GO_REF:00.
#   3 UniProt. A0A024RBG1   NUDT4B           NA        GO:004. GO_REF:00.
#   4 UniProt. A0A024RBG1   NUDT4B           NA        GO:005. GO_REF:00.
#   5 UniProt. A0A024RBG1   NUDT4B           NA        GO:005. GO_REF:00.
# # . with 606,830 more rows, and 11 more variables:
# #   evidence_code <fct>, with_or_from <chr>, aspect <fct>,
# #   db_object_name <chr>, db_object_synonym <chr>,
# #   db_object_type <fct>, taxon <fct>, date <date>,
# #   assigned_by <fct>, annotation_extension <chr>,
# #   gene_product_from_id <chr>
```

`go_annot_slim` *GO slim gene annotations*

Description

GO slims are subsets of the full GO which "give a broad overview of the ontology content without the detail of the specific fine grained terms". In order to annotate genes with GO slim terms, we take the annotations and search all ancestors of the terms up to the root of the ontology tree. From the ancestors we select the terms which are part of the slim subset.

Usage

```
go_annot_slim(
  organism = "human",
  slim = "generic",
  aspects = c("C", "F", "P"),
  cache = TRUE
)
```

Arguments

`organism`	Character: either "chicken", "cow", "dog", "human", "pig" or "uniprot_all".
`slim`	Character: the GO subset (GO slim) name. Available GO slims are: "agr" (Alliance for Genomics Resources), "generic", "aspergillus", "candida", "drosophila", "chembl", "metagenomic", "mouse", "plant", "pir" (Protein Information Resource), "pombe" and "yeast".
`aspects`	Character vector with some of the following elements: "C" (cellular component), "F" (molecular function) and "P" (biological process). Gene Ontology is three separate ontologies called as three aspects. By this parameter you can control which aspects to include in the output.
`cache`	Logical: Load the result from cache if available.

Details

Building the GO slim is resource intensive in its current implementation. For human annotation and generic GO slim it might take around 20 minutes. The result is saved into the cache so next time loading the data from there is really quick. If the `cache` option is `FALSE` the data will be built fresh (the annotation and ontology files still might come from cache), and the newly build GO slim will overwrite the cache instance.

Value

A tibble (data frame) of genes annotated with ontology terms in in the GO slim (subset).

See Also

- `go_annot_download`
- `go_ontology_download`
- `get_db`

Examples

```
## Not run:
goslim <- go_annot_slim(organism = 'human', slim = 'generic')
goslim
# # A tibble: 276,371 x 8
#    db     db_object_id db_object_symbol go_id aspect db_object_name
#    <fct>  <chr>        <chr>            <chr> <fct>  <chr>
#  1 UniPr. A0A024RBG1   NUDT4B           GO:0. F      Diphosphoinosito.
#  2 UniPr. A0A024RBG1   NUDT4B           GO:0. F      Diphosphoinosito.
#  3 UniPr. A0A024RBG1   NUDT4B           GO:0. C      Diphosphoinosito.
#  4 UniPr. A0A024RBG1   NUDT4B           GO:0. C      Diphosphoinosito.
#  5 UniPr. A0A024RBG1   NUDT4B           GO:0. C      Diphosphoinosito.
# # . with 276,366 more rows, and 2 more variables:
# #   db_object_synonym <chr>, db_object_type <fct>

## End(Not run)
```

`go_ontology_download` *The Gene Ontology tree*

Description

The Gene Ontology tree

Usage

```
go_ontology_download(
  basic = TRUE,
  tables = TRUE,
  subset = NULL,
 relations = c("is_a", "part_of", "occurs_in", "regulates", "positively_regulates",
    "negatively_regulates")
)
```

Arguments

`basic`	Logical: use the basic or the full version of GO. As written on the GO home page: "the basic version of the GO is filtered such that the graph is guaranteed to be acyclic and annotations can be propagated up the graph. The relations included are is a, part of, regulates, negatively regulates and positively regulates. This version excludes relationships that cross the 3 GO hierarchies. This version should be used with most GO-based annotation tools."
`tables`	In the result return data frames or nested lists. These later can be converted to each other if necessary. However converting from table to list is faster.
`subset`	Character: the GO subset (GO slim) name. GO slims are subsets of the full GO which "give a broad overview of the ontology content without the detail of the specific fine grained terms". This option, if not `NULL`, overrides the `basic` parameter. Available GO slims are: "agr" (Alliance for Genomics Resources), "generic", "aspergillus", "candida", "drosophila", "chembl", "metagenomic", "mouse", "plant", "pir" (Protein Information Resource), "pombe" and "yeast".
`relations`	Character vector: the relations to include in the processed data.

Value

A list with the following elements: 1) "names" a list with terms as names and names as values; 2) "namespaces" a list with terms as names and namespaces as values; 3) "relations" a list with relations between terms: terms are keys, values are lists with relations as names and character vectors of related terms as values; 4) "subsets" a list with terms as keys and character vectors of subset names as values (or `NULL` if the term does not belong to any subset); 5) "obsolete" character vector with all the terms labeled as obsolete. If the `tables` parameter is `TRUE`, "names", "namespaces", "relations" and "subsets" will be data frames (tibbles).

Examples

```
# retrieve the generic GO slim, a small subset of the full ontology
go <- go_ontology_download(subset = 'generic')
```

`graph_interaction` *Interaction data frame from igraph graph object*

Description

Convert an igraph graph object to interaction data frame. This is the reverse of the operation done by thje `interaction_graph` function. Networks can be easily converted to igraph objects, then you can make use of all igaph methods, and at the end, get back the interactions in a data frame, along with all new edge and node attributes.

Usage

```
graph_interaction(graph, implode = FALSE)
```

Arguments

`graph`	An igraph graph object created formerly from an OmniPath interactions data frame.
`implode`	Logical: restore the original state of the list type columns by imploding them to character vectors, subitems separated by semicolons.

Value

An interaction data frame.

See Also

`interaction_graph`

`guide2pharma_download` *Downloads interactions from the Guide to Pharmacology database*

Description

Downloads ligand-receptor interactions from the Guide to Pharmacology (IUPHAR/BPS) database (`https://www.guidetopharmacology.org/`).

Usage

```
guide2pharma_download()
```

Value

A tibble (data frame) of interactions as it is provided by the database

Examples

```
g2p_data <- guide2pharma_download()
g2p_data
# # A tibble: 21,586 x 38
#    target target_id target_gene_sym. target_uniprot target_ensembl_.
#    <chr>      <dbl> <chr>            <chr>          <chr>
#  1 12S-L.      1387 ALOX12           P18054         ENSG00000108839
#  2 15-LO.      1388 ALOX15           P16050         ENSG00000161905
#  3 15-LO.      1388 ALOX15           P16050         ENSG00000161905
#  4 15-LO.      1388 ALOX15           P16050         ENSG00000161905
# # . with 21,576 more rows, and 33 more variables: target_ligand <chr>,
# #   target_ligand_id <chr>, target_ligand_gene_symbol <chr>,
# ... (truncated)
```

`harmonizome_download` *Downloads a Harmonizome network dataset*

Description

Downloads a single network dataset from Harmonizome `https://maayanlab.cloud/Harmonizome`.

Usage

```
harmonizome_download(dataset)
```

Arguments

`dataset`	The dataset part of the URL. Please refer to the download section of the Harmonizome webpage.

Value

Data frame (tibble) with interactions.

Examples

```
harmonizome_data <- harmonizome_download('phosphositeplus')
harmonizome_data
# # A tibble: 6,013 x 7
#    source   source_desc source_id target target_desc target_id weight
#    <chr>    <chr>           <dbl> <chr>  <chr>           <dbl>  <dbl>
#  1 TP53     na               7157 STK17A na               9263      1
#  2 TP53     na               7157 TP53RK na             112858      1
#  3 TP53     na               7157 SMG1   na              23049      1
#  4 UPF1     na               5976 SMG1   na              23049      1
# # . with 6,003 more rows
```

`has_extra_attrs` *Tells if an interaction data frame has an extra_attrs column*

Description

Tells if an interaction data frame has an extra_attrs column

Usage

```
has_extra_attrs(data)
```

Arguments

`data`	An interaction data frame.

Value

Logical: TRUE if the data frame has the "extra_attrs" column.

See Also

- `extra_attrs`
- `extra_attrs_to_cols`
- `with_extra_attrs`
- `filter_extra_attrs`
- `extra_attr_values`

Examples

```
i <- omnipath(fields = "extra_attrs")
has_extra_attrs(i)
```

`hmdb_id_mapping_table` *Identifier translation table from HMDB*

Description

Identifier translation table from HMDB

Usage

```
hmdb_id_mapping_table(to, from, entity_type = "metabolite")
```

Arguments

`to`	Character or symbol: target ID type. See Details for possible values.
`from`	Character or symbol: source ID type. See Details for possible values.
`entity_type`	Character: "gene" and "smol" are short symbols for proteins, genes and small molecules respectively. Several other synonyms are also accepted.

Details

The arguments `to` and `from` can be provided either as character or as symbol (NSE). Their possible values are either HMDB XML tag names or synonyms listed at `id_types`.

Value

A data frame (tibble) with columns ‘From‘ and ‘To‘.

See Also

- `translate_ids`
- `id_types`
- `hmdb_table`
- `uniprot_full_id_mapping_table`
- `uniprot_id_mapping_table`
- `ensembl_id_mapping_table`
- `chalmers_gem_id_mapping_table`

Examples

```
hmdb_kegg <- hmdb_id_mapping_table("kegg", "hmdb")
hmdb_kegg
```

`hmdb_id_type` *HMDB identifier type label*

Description

HMDB identifier type label

Usage

```
hmdb_id_type(label)
```

Arguments

`label`	Character: an ID type label, as shown in the table at `translate_ids`

Value

Character: the HMDB specific ID type label, or the input unchanged if it could not be translated (still might be a valid identifier name). These labels should be valid HMDB field names, as used in HMDB XML files.

See Also

- `chalmers_gem_id_type`
- `uniprot_id_type`
- `ensembl_id_type`
- `uploadlists_id_type`

Examples

```
hmdb_id_type("hmdb")
# [1] "accession"
```

---

hmdb_metabolite_fields

Field names for the HMDB metabolite dataset

Description

Field names for the HMDB metabolite dataset

Usage

```
hmdb_metabolite_fields()
```

Value

Character vector of field names.

See Also

- `hmdb_table`
- `hmdb_protein_fields`

Examples

```
hmdb_metabolite_fields()
```

---

hmdb_protein_fields

Field names for the HMDB proteins dataset

Description

Field names for the HMDB proteins dataset

Usage

```
hmdb_protein_fields()
```

Value

Character vector of field names.

See Also

- `hmdb_table`
- `hmdb_metabolite_fields`

Examples

```
hmdb_protein_fields()
```

`hmdb_table` *Download a HMDB XML file and process it into a table*

Description

Download a HMDB XML file and process it into a table

Usage

```
hmdb_table(dataset = "metabolites", fields = NULL)
```

Arguments

`dataset`	Character: name of an HMDB XML dataset, such as "metabolites", "proteins", "urine", "serum", "csf", "saliva", "feces", "sweat".
`fields`	Character: fields to extract from the XML. This is a very minimal parser that is able to extract the text content of simple fields and multiple value fields which contain a list of leaves within one container tag under the record tag. A full list of fields available in HMDB is available by the `hmdb_protein_fields` and `hmdb_metabolite_fields` functions. By default, all fields available in the dataset are extracted.

Value

A data frame (tibble) with each column corresponding to a field.

See Also

- `hmdb_protein_fields`
- `hmdb_metabolite_fields`

Examples

```
hmdb_table()
```

`homologene_download` *Orthology table for a pair of organisms*

Description

Orthologous pairs of genes for a pair of organisms from NCBI HomoloGene, using one identifier type.

Usage

```
homologene_download(
  target = 10090L,
  source = 9606L,
  id_type = "genesymbol",
  hgroup_size = FALSE
)
```

Arguments

`target`	Character or integer: name or ID of the target organism.
`source`	Character or integer: name or ID of the source organism.
`id_type`	Symbol or character: identifier type, possible values are "genesymbol", "entrez", "refseqp" or "gi".
`hgroup_size`	Logical: include a column with the size of the homology groups. This column distinguishes one-to-one and one-to-many or many-to-many mappings.

Details

The operation of this function is symmetric, *source* and *target* are interchangeable but determine the column layout of the output. The column "hgroup" is a numberic identifier of the homology groups. Most of the groups consist of one pair of orthologous genes (one-to-one mapping), and a few of them multiple ones (one-to-many or many-to-many mappings).

Value

A data frame with orthologous identifiers between the two organisms.

See Also

- `homologene_raw`
- `homologene_uniprot_orthology`

Examples

```
chimp_human <- homologene_download(chimpanzee, human, refseqp)
chimp_human
# # A tibble: 17,737 × 3
#    hgroup refseqp_source refseqp_target
#     <int> <chr>          <chr>
#  1      3 NP_000007.1    NP_001104286.1
#  2      5 NP_000009.1    XP_003315394.1
```

```
#  3      6 NP_000010.1    XP_508738.2
#  4      7 NP_001096.1    XP_001145316.1
#  5      9 NP_000014.1    XP_523792.2
# # . with 17,732 more rows
```

`homologene_organisms` *Organisms in NCBI HomoloGene*

Description

Organisms in NCBI HomoloGene

Usage

```
homologene_organisms(name_type = "ncbi")
```

Arguments

`name_type`	Character: type of the returned name or identifier. Many synonyms are accepted, the shortest ones: "latin", "ncbi", "common", "ensembl". Case unsensitive.

Details

Not all NCBI Taxonomy IDs can be translated to common or latin names. It means some organisms will be missing if translated to those name types. In the future we will address this issue, until then if you want to see all organisms use NCBI Taxonomy IDs.

Value

A character vector of organism names.

`homologene_raw` *Orthology data from NCBI HomoloGene*

Description

Retrieves NCBI HomoloGene data without any processing. Processed tables are more useful for most purposes, see below other functions that provide those. Genes of various organisms are grouped into homology groups ("hgroup" column). Organisms are identified by NCBI Taxonomy IDs, genes are identified by four different identifier types.

Usage

```
homologene_raw()
```

Value

A data frame as provided by NCBI HomoloGene.

See Also

- homologene_download

Examples

```
hg <- homologene_raw()
hg
# # A tibble: 275,237 × 6
#    hgroup ncbi_taxid entrez  genesymbol  gi        refseqp
#     <int>      <int> <chr>   <chr>       <chr>     <chr>
#   1      3       9606 34      ACADM       4557231   NP_000007.1
#   2      3       9598 469356  ACADM       160961497 NP_001104286.1
#   3      3       9544 705168  ACADM       109008502 XP_001101274.1
#   4      3       9615 490207  ACADM       545503811 XP_005622188.1
#   5      3       9913 505968  ACADM       115497690 NP_001068703.1
# # . with 275,232 more rows

# which organisms are available?
common_name(unique(hg$ncbi_taxid))
#  [1] "Human" "Chimpanzee" "Macaque" "Dog" "Cow" "Mouse" "Rat" "Zebrafish"
#  [9] "D. melanogaster" "Caenorhabditis elegans (PRJNA13758)"
# [11] "Tropical clawed frog" "Chicken"
# ...and 9 more organisms with missing English names.
```

---

homologene_uniprot_orthology

Orthology table with UniProt IDs

Description

Orthologous pairs of UniProt IDs for a pair of organisms, based on NCBI HomoloGene data.

Usage

```
homologene_uniprot_orthology(target = 10090L, source = 9606L, by = entrez, ...)
```

Arguments

`target`	Character or integer: name or ID of the target organism.
`source`	Character or integer: name or ID of the source organism.
`by`	Symbol or character: the identifier type in NCBI HomoloGene to use. Possible values are "refseqp", "entrez", "genesymbol", "gi".
`...`	Further arguments passed to `translate_ids`.

Value

A data frame with orthologous pairs of UniProt IDs.

Examples

```
homologene_uniprot_orthology(by = genesymbol)
# # A tibble: 14,235 × 2
#    source target
#    <chr>  <chr>
#  1 P11310 P45952
#  2 P49748 P50544
#  3 P24752 Q8QZT1
#  4 Q04771 P37172
#  5 Q16586 P82350
# # . with 14,230 more rows
```

`hpo_download` *Downloads protein annotations from Human Phenotype Ontology*

Description

Human Phenotype Ontology (HPO) provides a standardized vocabulary of phenotypic abnormalities encountered in human disease. Each term in the HPO describes a phenotypic abnormality. HPO currently contains over 13,000 terms and over 156,000 annotations to hereditary diseases. See more at `https://hpo.jax.org/app/`.

Usage

```
hpo_download()
```

Value

A tibble (data frame) of annotations as it is provided by the database

Examples

```
hpo_data <- hpo_download()
hpo_data
# # A tibble: 231,738 x 9
#    entrez_gene_id entrez_gene_symb. hpo_term_id hpo_term_name
#             <dbl> <chr>             <chr>       <chr>
#  1           8192 CLPP              HP:0000013  Hypoplasia of the ute.
#  2           8192 CLPP              HP:0004322  Short stature
#  3           8192 CLPP              HP:0000786  Primary amenorrhea
#  4           8192 CLPP              HP:0000007  Autosomal recessive i.
#  5           8192 CLPP              HP:0000815  Hypergonadotropic hyp.
# # . with 231,733 more rows, and 5 more variables:
# #   frequency_raw <chr>, frequency_hpo <chr>, info_gd_source <chr>,
# #   gd_source <chr>, disease_id <chr>
```

`htridb_download` *Downloads TF-target interactions from HTRIdb*

Description

HTRIdb (`https://www.lbbc.ibb.unesp.br/htri/`) is a database of literature curated human TF-target interactions. As the database is recently offline, the data is distributed by the OmniPath rescued data repository (`https://rescued.omnipathdb.org/`).

Usage

```
htridb_download()
```

Value

Data frame (tibble) with interactions.

Examples

```
htridb_data <- htridb_download()
htridb_data
# # A tibble: 18,630 x 7
#      OID GENEID_TF SYMBOL_TF GENEID_TG SYMBOL_TG TECHNIQUE
#    <dbl>     <dbl> <chr>         <dbl> <chr>     <chr>
#  1 32399       142 PARP1           675 BRCA2     Electrophoretic Mobi.
#  2 32399       142 PARP1           675 BRCA2     Chromatin Immunoprec.
#  3 28907       196 AHR            1543 CYP1A1    Chromatin Immunoprec.
#  4 29466       196 AHR            1543 CYP1A1    Electrophoretic Mobi.
#  5 28911       196 AHR            1543 CYP1A1    Chromatin Immunoprec.
# # . with 18,620 more rows, and 1 more variable: PUBMED_ID <chr>
```

`id_translation_resources` *List available ID translation resources*

Description

List available ID translation resources

Usage

```
id_translation_resources()
```

Value

A character vector with the names of the available ID translation resources.

Examples

```
id_translation_resources()
```

`id_types` *ID types and synonyms in identifier translation*

Description

ID types and synonyms in identifier translation

Usage

```
id_types()
```

Value

Data frame with 4 columns: the ID type labels in the resource, their synonyms in OmniPath (this package), the name of the ID translation resource, and the entity type.

See Also

- `translate_ids`
- `translate_ids_multi`
- `ensembl_id_mapping_table`
- `uniprot_id_mapping_table`
- `hmdb_id_mapping_table`
- `chalmers_gem_id_mapping_table`
- `uniprot_full_id_mapping_table`
- `ensembl_id_type`
- `uniprot_id_type`
- `hmdb_id_type`
- `chalmers_gem_id_type`

Examples

```
id_types()
```

`inbiomap_download` *Downloads and preprocesses network data from InWeb InBioMap*

Description

Downloads the data by `inbiomap_raw`, extracts the UniProt IDs, Gene Symbols and scores and removes the irrelevant columns.

Usage

```
inbiomap_download(...)
```

Arguments

`...`	Passed to `inbiomap_raw`.

Value

A data frame (tibble) of interactions.

See Also

`inbiomap_raw`

Examples

```
## Not run:
inbiomap_interactions <- inbiomap_download()
inbiomap_interactions

## End(Not run)
# # A tibble: 625,641 x 7
#    uniprot_a uniprot_b genesymbol_a genesymbol_b inferred score1 score2
#    <chr>     <chr>     <chr>        <chr>        <lgl>     <dbl>  <dbl>
#  1 A0A5B9    P01892    TRBC2        HLA-A        FALSE     0.417 0.458
#  2 A0AUZ9    Q96CV9    KANSL1L      OPTN         FALSE     0.155 0.0761
#  3 A0AV02    P24941    SLC12A8      CDK2         TRUE      0.156 0.0783
#  4 A0AV02    Q00526    SLC12A8      CDK3         TRUE      0.157 0.0821
#  5 A0AV96    P0CG48    RBM47        UBC          FALSE     0.144 0.0494
# # . with 625,631 more rows
```

`inbiomap_raw` *Downloads network data from InWeb InBioMap*

Description

Downloads the data from `https://inbio-discover.com/map.html#downloads` in tar.gz format, extracts the PSI MITAB table and returns it as a data frame.

Usage

```
inbiomap_raw(curl_verbose = FALSE)
```

Arguments

`curl_verbose`	Logical. Perform CURL requests in verbose mode for debugging purposes.

Value

A data frame (tibble) with the extracted interaction table.

See Also

`inbiomap_download`

Examples

```
## Not run:
inbiomap_psimitab <- inbiomap_raw()

## End(Not run)
```

`interaction_datasets` *Datasets in the OmniPath Interactions database*

Description

Datasets in the OmniPath Interactions database

Usage

```
interaction_datasets()
```

Value

Character: labels of interaction datasets.

Examples

```
interaction_datasets()
```

`interaction_graph` *Build Omnipath interaction graph*

Description

Transforms the interactions data frame to an igraph graph object.

Usage

```
interaction_graph(interactions = interactions)
```

Arguments

`interactions`	data.frame created by • `enzyme_substrate` • `omnipath-interactions`

Value

An igraph graph object.

See Also

- graph_interaction
- import_omnipath_interactions
- import_pathwayextra_interactions
- import_kinaseextra_interactions
- import_ligrecextra_interactions
- import_dorothea_interactions
- import_mirnatarget_interactions
- import_all_interactions
- giant_component
- find_all_paths

Examples

```
interactions <- import_omnipath_interactions(resources = c('SignaLink3'))
g <- interaction_graph(interactions)
```

`interaction_resources` *Interaction resources available in Omnipath*

Description

Names of the resources available in https://omnipathdb.org/interactions.

Usage

```
interaction_resources(dataset = NULL)
```

Arguments

`dataset`	a dataset within the interactions query type. Currently available datasets are ‘omnipath‘, ‘kinaseextra‘, ‘pathwayextra‘, ‘ligrecextra‘, ‘collectri‘, ‘dorothea‘, ‘tf_target‘, ‘tf_mirna‘, ‘mirnatarget‘, ‘lncrna_mrna‘ and ‘small_molecule_protein‘.

Value

Character: names of the interaction resources.

See Also

- resources
- omnipath
- pathwayextra
- kinaseextra
- ligrecextra
- post_translational

- `dorothea`
- `collectri`
- `tf_target`
- `transcriptional`
- `mirna_target`
- `tf_mirna`
- `small_molecule`
- `all_interactions`

Examples

```
interaction_resources()
```

`interaction_types` *Interaction types in the OmniPath Interactions database*

Description

Interaction types in the OmniPath Interactions database

Usage

```
interaction_types()
```

Value

Character: labels of interaction types.

Examples

```
interaction_types()
```

`intercell` *Cell-cell communication roles from OmniPath*

Description

Roles of proteins in intercellular communication from the `https://omnipathdb.org/intercell` endpoint of the OmniPath web service. It provides information on the roles in inter-cellular signaling. E.g. if a protein is a ligand, a receptor, an extracellular matrix (ECM) component, etc.

Usage

```
intercell(
  categories = NULL,
  parent = NULL,
  scope = NULL,
  aspect = NULL,
  source = NULL,
  transmitter = NULL,
  receiver = NULL,
  secreted = NULL,
  plasma_membrane_peripheral = NULL,
  plasma_membrane_transmembrane = NULL,
  proteins = NULL,
  topology = NULL,
  causality = NULL,
  consensus_percentile = NULL,
  loc_consensus_percentile = NULL,
  ...
)
```

Arguments

`categories`	vector containing the categories to be retrieved. All the genes belonging to those categories will be returned. For further information about the categories see `get_intercell_categories`.
`parent`	vector containing the parent classes to be retrieved. All the genes belonging to those classes will be returned. For furter information about the main classes see `get_intercell_categories`.
`scope`	either ‘specific‘ or ‘generic‘
`aspect`	either ‘locational‘ or ‘functional‘
`source`	either ‘resource_specific‘ or ‘composite‘
`transmitter`	logical, include only transmitters i.e. proteins delivering signal from a cell to its environment.
`receiver`	logical, include only receivers i.e. proteins delivering signal to the cell from its environment.
`secreted`	logical, include only secreted proteins
`plasma_membrane_peripheral`	logical, include only plasma membrane peripheral membrane proteins.
`plasma_membrane_transmembrane`	logical, include only plasma membrane transmembrane proteins.
`proteins`	limit the query to certain proteins
`topology`	topology categories: one or more of ‘secreted‘ (sec), ‘plasma_membrane_peripheral‘ (pmp), ‘plasma_membrane_transmembrane‘ (pmtm) (both short or long notation can be used).
`causality`	‘transmitter‘ (trans), ‘receiver‘ (rec) or ‘both‘ (both short or long notation can be used).
`consensus_percentile`	Numeric: a percentile cut off for the consensus score of generic categories. The consensus score is the number of resources supporting the classification of an

entity into a category based on combined information of many resources. Here you can apply a cut-off, keeping only the annotations supported by a higher number of resources than a certain percentile of each category. If `NULL` no filtering will be performed. The value is either in the 0-1 range, or will be divided by 100 if greater than 1. The percentiles will be calculated against the generic composite categories and then will be applied to their resource specific annotations and specific child categories.

`loc_consensus_percentile`

Numeric: similar to `consensus_percentile` for major localizations. For example, with a value of 50, the secreted, plasma membrane transmembrane or peripheral attributes will be true only where at least 50 percent of the resources support these.

`...` Arguments passed on to `omnipath_query`

`organism` Character or integer: name or NCBI Taxonomy ID of the organism. OmniPath is built of human data, and the web service provides orthology translated interactions and enzyme-substrate relationships for mouse and rat. For other organisms and query types, orthology translation will be called automatically on the downloaded human data before returning the result.

`resources` Character vector: name of one or more resources. Restrict the data to these resources. For a complete list of available resources, call the ‘<query_type>_resources‘ functions for the query type of interst.

`fields` Character vector: additional fields to include in the result. For a list of available fields, call ‘query_info("interactions")‘.

`default_fields` Logical: if TRUE, the default fields will be included.

`silent` Logical: if TRUE, no messages will be printed. By default a summary message is printed upon successful download.

`logicals` Character vector: fields to be cast to logical.

`format` Character: if "json", JSON will be retrieved and processed into a nested list; any other value will return data frame.

`download_args` List: parameters to pass to the download function, which is ‘readr::read_tsv‘ by default, and ‘jsonlite::safe_load‘.

`license` Character: license restrictions. By default, data from resources allowing "academic" use is returned by OmniPath. If you use the data for work in a company, you can provide "commercial" or "for-profit", which will restrict the data to those records which are supported by resources that allow for-profit use.

`password` Character: password for the OmniPath web service. You can provide a special password here which enables the use of ‘license = "ignore"‘ option, completely bypassing the license filter.

`exclude` Character vector: resource or dataset names to be excluded. The data will be filtered after download to remove records of the excluded datasets and resources.

`json_param` List: parameters to pass to the ‘jsonlite::fromJSON‘ when processing JSON columns embedded in the downloaded data. Such columns are "extra_attrs" and "evidences". These are optional columns which provide a lot of extra details about interactions.

`strict_evidences` Logical: reconstruct the "sources" and "references" columns of interaction data frames based on the "evidences" column, strictly filtering them to the queried datasets and resources. Without this, the "sources" and "references" fields for each record might contain information for datasets and resources other than the queried ones, because the downloaded records are a result of a simple filtering of an already integrated data frame.

`genesymbol_resource` Character: "uniprot" (default) or "ensembl". The OmniPath web service uses the primary gene symbols as provided by UniProt. By passing "ensembl" here, the UniProt gene symbols will be replaced by the ones used in Ensembl. This translation results in a loss of a few records, and multiplication of another few records due to ambiguous translation.

`cache` Logical: use caching, load data from and save to the. The cache directory by default belongs to the user, located in the user's default cache directory, and named "OmnipathR". Find out about it by `getOption("omnipathr.cachedir")`. Can be changed by `omnipath_set_cachedir`.

Value

A data frame of intercellular communication roles.

See Also

- `intercell_network`
- `intercell_consensus_filter`
- `filter_intercell`
- `intercell_categories`
- `intercell_generic_categories`
- `intercell_resources`
- `intercell_summary`
- `intercell_network`

Examples

```
ecm_proteins <- intercell(categories = "ecm")
```

`intercell_categories` *Categories in the intercell database of OmniPath*

Description

Retrieves a list of categories from `https://omnipathdb.org/intercell`.

Usage

```
intercell_categories()
```

Value

character vector with the different intercell categories

See Also

- `intercell`
- `intercell_generic_categories`
- `intercell_summary`

Examples

```
intercell_categories()
```

`intercell_consensus_filter`

Quality filter for intercell annotations

Description

Quality filter for intercell annotations

Usage

```
intercell_consensus_filter(
  data,
  percentile = NULL,
  loc_percentile = NULL,
  topology = NULL
)
```

Arguments

`data`	A data frame with intercell annotations, as provided by `intercell`.
`percentile`	Numeric: a percentile cut off for the consensus score of composite categories. The consensus score is the number of resources supporting the classification of an entity into a category based on combined information of many resources. Here you can apply a cut-off, keeping only the annotations supported by a higher number of resources than a certain percentile of each category. If `NULL` no filtering will be performed. The value is either in the 0-1 range, or will be divided by 100 if greater than 1. The percentiles will be calculated against the generic composite categories and then will be applied to their resource specific annotations and specific child categories.
`loc_percentile`	Numeric: similar to `percentile` for major localizations. For example, with a value of 50, the secreted, plasma membrane transmembrane or peripheral attributes will be `TRUE` only where at least 50 percent of the resources support these.
`topology`	Character vector: list of allowed topologies, possible values are *"secreted"*, *"plasma_membrane_peripheral"* and *"plasma_membrane_transmembrane"*.

Value

The data frame in `data` filtered by the consensus scores.

See Also

- `resources`
- `intercell`
- `filter_intercell`
- `intercell_categories`
- `intercell_generic_categories`
- `intercell_resources`
- `intercell_summary`
- `intercell_network`

Examples

```
ligand_receptor <- intercell(parent = c("ligand", "receptor"))
nrow(ligand_receptor)
# [1] 50174
lr_q50 <- intercell_consensus_filter(ligand_receptor, 50)
nrow(lr_q50)
# [1] 42863
```

---

intercell_generic_categories

Retrieves a list of the generic categories in the intercell database of OmniPath

---

Description

Retrieves a list of the generic categories from `https://omnipathdb.org/intercell`.

Usage

```
intercell_generic_categories()
```

Value

character vector with the different intercell main classes

See Also

- `intercell`
- `intercell_categories`
- `intercell_summary`

Examples

```
intercell_generic_categories()
```

`intercell_network` *Intercellular communication network*

Description

Imports an intercellular network by combining intercellular annotations and protein interactions. First imports a network of protein-protein interactions. Then, it retrieves annotations about the proteins intercellular communication roles, once for the transmitter (delivering information from the expressing cell) and second, the receiver (receiving signal and relaying it towards the expressing cell) side. These 3 queries can be customized by providing parameters in lists which will be passed to the respective methods (`omnipath_interactions` for the network and `intercell` for the annotations). Finally the 3 data frames combined in a way that the source proteins in each interaction annotated by the transmitter, and the target proteins by the receiver categories. If undirected interactions present (these are disabled by default) they will be duplicated, i.e. both partners can be both receiver and transmitter.

Usage

```
intercell_network(
  interactions_param = list(),
  transmitter_param = list(),
  receiver_param = list(),
  resources = NULL,
  entity_types = NULL,
  ligand_receptor = FALSE,
  high_confidence = FALSE,
  simplify = FALSE,
  unique_pairs = FALSE,
  consensus_percentile = NULL,
  loc_consensus_percentile = NULL,
  omnipath = TRUE,
  ligrecextra = TRUE,
  kinaseextra = !high_confidence,
  pathwayextra = !high_confidence,
  ...
)
```

Arguments

`interactions_param`	a list with arguments for an interactions query; `omnipath-interactions`.
`transmitter_param`	a list with arguments for `intercell`, to define the transmitter side of intercellular connections
`receiver_param`	a list with arguments for `intercell`, to define the receiver side of intercellular connections
`resources`	A character vector of resources to be applied to both the interactions and the annotations. For example, `resources = 'CellChatDB'` will download the transmitters and receivers defined by CellChatDB, connected by connections from CellChatDB.

`entity_types`	Character, possible values are "protein", "complex" or both.
`ligand_receptor`	Logical. If `TRUE`, only *ligand* and *receptor* annotations will be used instead of the more generic *transmitter* and *receiver* categories.
`high_confidence`	Logical: shortcut to do some filtering in order to include only higher confidence interactions. The intercell database of OmniPath covers a very broad range of possible ways of cell to cell communication, and the pieces of information, such as localization, topology, function and interaction, are combined from many, often independent sources. This unavoidably result some weird and unexpected combinations which are false positives in the context of intercellular communication. This option sets some minimum criteria to remove most (but definitely not all!) of the wrong connections. These criteria are the followings: 1) the receiver must be plasma membrane transmembrane; 2) the curation effort for interactions must be larger than one; 3) the consensus score for annotations must be larger than the 50 percentile within the generic category (you can override this by `consensus_percentile`). 4) the transmitter must be secreted or exposed on the plasma membrane. 5) The major localizations have to be supported by at least 30 percent of the relevant resources (you can override this by `loc_consensus_percentile`). 6) The datasets with lower level of curation (*kinaseextra* and *pathwayextra*) will be disabled. These criteria are of medium stringency, you can always tune them to be more relaxed or stringent by filtering manually, using `filter_intercell_network`.
`simplify`	Logical: keep only the most often used columns. This function combines a network data frame with two copies of the intercell annotation data frames, all of them already having quite some columns. With this option we keep only the names of the interacting pair, their intercellular communication roles, and the minimal information of the origin of both the interaction and the annotations.
`unique_pairs`	Logical: instead of having separate rows for each pair of annotations, drop the annotations and reduce the data frame to unique interacting pairs. See `unique_intercell_network` for details.
`consensus_percentile`	Numeric: a percentile cut off for the consensus score of generic categories in intercell annotations. The consensus score is the number of resources supporting the classification of an entity into a category based on combined information of many resources. Here you can apply a cut-off, keeping only the annotations supported by a higher number of resources than a certain percentile of each category. If `NULL` no filtering will be performed. The value is either in the 0-1 range, or will be divided by 100 if greater than 1. The percentiles will be calculated against the generic composite categories and then will be applied to their resource specific annotations and specific child categories.
`loc_consensus_percentile`	Numeric: similar to `consensus_percentile` for major localizations. For example, with a value of 50, the secreted, plasma membrane transmembrane or peripheral attributes will be `TRUE` only where at least 50 percent of the resources support these.
`omnipath`	Logical: shortcut to include the *omnipath* dataset in the interactions query.
`ligrecextra`	Logical: shortcut to include the *ligrecextra* dataset in the interactions query.
`kinaseextra`	Logical: shortcut to include the *kinaseextra* dataset in the interactions query.
`pathwayextra`	Logical: shortcut to include the *pathwayextra* dataset in the interactions query.

`...`	If `simplify` or `unique_pairs` is `TRUE`, additional column names can be passed here to `dplyr::select` on the final data frame. Otherwise ignored.

Details

By default this function creates almost the largest possible network of intercellular interactions. However, this might contain a large number of false positives. Please refer to the documentation of the arguments, especially `high_confidence`, and the `filter_intercell_network` function. Note: if you restrict the query to certain intercell annotation resources or small categories, it's not recommended to use the `consensus_percentile` or `high_confidence` options, instead filter the network with `filter_intercell_network` for more consistent results.

Value

A dataframe containing information about protein-protein interactions and the inter-cellular roles of the protiens involved in those interactions.

See Also

- `intercell`
- `intercell_summary`
- `intercell_categories`
- `intercell_generic_categories`
- `intercell`
- `omnipath`
- `pathwayextra`
- `kinaseextra`
- `ligrecextra`
- `unique_intercell_network`
- `simplify_intercell_network`
- `filter_intercell_network`

Examples

```
intercell_network <- intercell_network(
    interactions_param = list(datasets = 'ligrecextra'),
    receiver_param = list(categories = c('receptor', 'transporter')),
    transmitter_param = list(categories = c('ligand', 'secreted_enzyme'))
)
```

`intercell_resources` *Retrieves a list of intercellular communication resources available in OmniPath*

Description

Retrieves a list of the databases from `https://omnipathdb.org/intercell`.

Usage

```
intercell_resources(dataset = NULL)
```

Arguments

`dataset`	ignored at this query type

Value

character vector with the names of the databases

See Also

- `resources`
- `intercell`
- `filter_intercell`
- `intercell_categories`
- `intercell_generic_categories`
- `intercell_summary`
- `intercell_network`

Examples

```
intercell_resources()
```

`intercell_summary` *Full list of intercell categories and resources*

Description

Full list of intercell categories and resources

Usage

```
intercell_summary()
```

Value

A data frame of categories and resources.

Examples

```
ic_cat <- intercell_categories()
ic_cat
# # A tibble: 1,125 x 3
#    category               parent                 database
#    <chr>                  <chr>                  <chr>
#  1 transmembrane          transmembrane          UniProt_location
#  2 transmembrane          transmembrane          UniProt_topology
#  3 transmembrane          transmembrane          UniProt_keyword
#  4 transmembrane          transmembrane_predicted Phobius
#  5 transmembrane_phobius  transmembrane_predicted Almen2009
# # . with 1,120 more rows
```

`is_ontology_id` *Looks like an ontology ID*

Description

Tells if the input has the typical format of ontology IDs, i.e. a code of capital letters, a colon, followed by a numeric code.

Usage

```
is_ontology_id(terms)
```

Arguments

`terms`	Character vector with strings to check.

Value

A logical vector with the same length as the input.

Examples

```
is_ontology_id(c('GO:0000001', 'reproduction'))
# [1]  TRUE FALSE
```

`is_swissprot` *Check for SwissProt IDs*

Description

Check for SwissProt IDs

Usage

```
is_swissprot(uniprots, organism = 9606)
```

Arguments

`uniprots`	Character vector of UniProt IDs.
`organism`	Character or integer: name or identifier of the organism.

Value

Logical vector TRUE for SwissProt IDs and FALSE for any other element.

Examples

```
is_swissprot(c("Q05BL1", "A0A654IBU3", "P00533"))
# [1] FALSE FALSE TRUE
```

`is_trembl` *Check for TrEMBL IDs*

Description

Check for TrEMBL IDs

Usage

```
is_trembl(uniprots, organism = 9606)
```

Arguments

`uniprots`	Character vector of UniProt IDs.
`organism`	Character or integer: name or identifier of the organism.

Value

Logical vector TRUE for TrEMBL IDs and FALSE for any other element.

Examples

```
is_trembl(c("Q05BL1", "A0A654IBU3", "P00533"))
# [1] TRUE TRUE FALSE
```

`is_uniprot` *Looks like a UniProt ID?*

Description

This function checks only the format of the IDs, no guarantee that these IDs exist in UniProt.

Usage

```
is_uniprot(identifiers)
```

Arguments

`identifiers`	Character: one or more identifiers (typically a single string, a vector or a data frame column).

Value

Logical: true if all elements in the input (except NAs) looks like valid UniProt IDs. If the input is not a character vector, ‘FALSE‘ is returned.

Examples

```
is_uniprot(all_uniprot_acs())
# [1] TRUE
is_uniprot("P00533")
# [1] TRUE
is_uniprot("pizza")
# [1] FALSE
```

`kegg_info` *Information about a KEGG Pathway*

Description

Information about a KEGG Pathway

Usage

```
kegg_info(pathway_id)
```

Arguments

`pathway_id`	Character: a KEGG Pathway identifier, e.g. "hsa04710". For a complete list of IDs see `kegg_pathway_list`.

Value

List with the pathway information.

See Also

- `kegg_pathway_list`
- `kegg_picture`
- `kegg_open`

Examples

```
kegg_info('map00563')
```

`kegg_open` *Open a KEGG Pathway diagram in the browser*

Description

Open a KEGG Pathway diagram in the browser

Usage

```
kegg_open(pathway_id)
```

Arguments

`pathway_id`	Character: a KEGG Pathway identifier, e.g. "hsa04710". For a complete list of IDs see `kegg_pathway_list`.

Details

To open URLs in the web browser the "browser" option must to be set to a a valid executable. You can check the value of this option by `getOption("browser")`. If your browser is firefox and the executable is located in the system path, you can set the option to point to it: `options(browser = "firefox")`. To make it a permanent setting, you can also include this in your `.Rprofile` file.

Value

Returns `NULL`.

See Also

- `kegg_pathway_list`
- `kegg_picture`
- `kegg_info`

Examples

```
if(any(getOption('browser') != '')) kegg_open('hsa04710')
```

`kegg_pathways_download`

Download the KEGG Pathways database

Description

Downloads all pathway diagrams in the KEGG Pathways database in KGML format and processes the XML to extract the interactions.

Usage

```
kegg_pathways_download(max_expansion = NULL, simplify = FALSE)
```

Arguments

`max_expansion`	Numeric: the maximum number of relations derived from a single relation record. As one entry might represent more than one molecular entities, one relation might yield a large number of relations in the processing. This happens in a combinatorial way, e.g. if the two entries represent 3 and 4 entities, that results 12 relations. If `NULL`, all relations will be expanded.
`simplify`	Logical: remove KEGG's internal identifiers and the pathway annotations, keep only unique interactions with direction and effect sign.

Value

A data frame (tibble) of interactions.

See Also

- `kegg_pathway_list`
- `kegg_process`
- `kegg_pathway_download`

Examples

```
## Not run:
kegg_pw <- kegg_pathways_download(simplify = TRUE)
kegg_pw
# # A tibble: 6,765 x 6
#    uniprot_source uniprot_target type  effect genesymbol_source
#    <chr>          <chr>          <chr> <chr>  <chr>
#  1 Q03113         Q15283         PPrel activ. GNA12
#  2 Q9Y4G8         P62070         PPrel activ. RAPGEF2
#  3 Q13972         P62070         PPrel activ. RASGRF1
#  4 O95267         P62070         PPrel activ. RASGRP1
#  5 P62834         P15056         PPrel activ. RAP1A
# # . with 6,760 more rows, and 1 more variable: genesymbol_target <chr>

## End(Not run)
```

`kegg_pathway_annotations` *Protein pathway annotations*

Description

Downloads all KEGG pathways and creates a table of protein-pathway annotations.

Usage

```
kegg_pathway_annotations(pathways = NULL)
```

Arguments

`pathways`	A table of KEGG pathways as produced by `kegg_pathways_download`.

Value

A data frame (tibble) with UniProt IDs and pathway names.

See Also

`kegg_pathways_download`

Examples

```
## Not run:
kegg_pw_annot <- kegg_pathway_annotations()
kegg_pw_annot
# # A tibble: 7,341 x 4
#    uniprot genesymbol pathway                pathway_id
#    <chr>   <chr>      <chr>                  <chr>
#  1 Q03113  GNA12      MAPK signaling pathway hsa04010
#  2 Q9Y4G8  RAPGEF2    MAPK signaling pathway hsa04010
#  3 Q13972  RASGRF1    MAPK signaling pathway hsa04010
#  4 O95267  RASGRP1    MAPK signaling pathway hsa04010
#  5 P62834  RAP1A      MAPK signaling pathway hsa04010
# # . with 7,336 more rows

## End(Not run)
```

`kegg_pathway_download` *Download one KEGG pathway*

Description

Downloads one pathway diagram from the KEGG Pathways database in KGML format and processes the XML to extract the interactions.

Usage

```
kegg_pathway_download(
  pathway_id,
  process = TRUE,
  max_expansion = NULL,
  simplify = FALSE
)
```

Arguments

`pathway_id`	Character: a KEGG pathway identifier, for example "hsa04350".
`process`	Logical: process the data or return it in raw format. processing means joining the entries and relations into a single data frame and adding UniProt IDs.
`max_expansion`	Numeric: the maximum number of relations derived from a single relation record. As one entry might represent more than one molecular entities, one relation might yield a large number of relations in the processing. This happens in a combinatorial way, e.g. if the two entries represent 3 and 4 entities, that results 12 relations. If `NULL`, all relations will be expanded.
`simplify`	Logical: remove KEGG's internal identifiers and the pathway annotations, keep only unique interactions with direction and effect sign.

Value

A data frame (tibble) of interactions if `process` is `TRUE`, otherwise a list with two data frames: "entries" is a raw table of the entries while "relations" is a table of relations extracted from the KGML file.

See Also

- `kegg_process`
- `kegg_pathways_download`
- `kegg_pathway_list`

Examples

```
tgf_pathway <- kegg_pathway_download('hsa04350')
tgf_pathway
# # A tibble: 50 x 12
#    source target type  effect arrow relation_id kegg_id_source
#    <chr>  <chr>  <chr> <chr>  <chr> <chr>       <chr>
#  1 51     49     PPrel activ. -->   hsa04350:1  hsa:7040 hsa:.
#  2 57     55     PPrel activ. -->   hsa04350:2  hsa:151449 hs.
#  3 34     32     PPrel activ. -->   hsa04350:3  hsa:3624 hsa:.
#  4 20     17     PPrel activ. -->   hsa04350:4  hsa:4838
#  5 60     46     PPrel activ. -->   hsa04350:5  hsa:4086 hsa:.
# # . with 45 more rows, and 5 more variables: genesymbol_source <chr>,
# #   uniprot_source <chr>, kegg_id_target <chr>,
# #   genesymbol_target <chr>, uniprot_target <chr>
```

`kegg_pathway_list` *List of KEGG pathways*

Description

Retrieves a list of available KEGG pathways.

Usage

```
kegg_pathway_list()
```

Value

Data frame of pathway names and identifiers.

See Also

- `kegg_process`
- `kegg_pathway_download`
- `kegg_pathways_download`
- `kegg_open`
- `kegg_picture`
- `kegg_info`

Examples

```
kegg_pws <- kegg_pathway_list()
kegg_pws
# # A tibble: 521 x 2
#    id       name
#    <chr>    <chr>
#  1 map01100 Metabolic pathways
#  2 map01110 Biosynthesis of secondary metabolites
#  3 map01120 Microbial metabolism in diverse environments
#  4 map01200 Carbon metabolism
#  5 map01210 2-Oxocarboxylic acid metabolism
#  6 map01212 Fatty acid metabolism
#  7 map01230 Biosynthesis of amino acids
# # . with 514 more rows
```

`kegg_picture` *Download a pathway diagram as a picture*

Description

Downloads a KEGG Pathway diagram as a PNG image.

Usage

```
kegg_picture(pathway_id, path = NULL)
```

Arguments

`pathway_id`	Character: a KEGG Pathway identifier, e.g. "hsa04710". For a complete list of IDs see `kegg_pathway_list`.
`path`	Character: save the image to this path. If `NULL`, the image will be saved in the current directory under the name `<pathway_id>.png`.

Value

Invisibly returns the path to the downloaded file.

See Also

`kegg_pathway_list`

- `kegg_pathway_list`
- `kegg_open`
- `kegg_info`

Examples

```
kegg_picture('hsa04710')
kegg_picture('hsa04710', path = 'foo/bar')
kegg_picture('hsa04710', path = 'foo/bar/circadian.png')
```

`kegg_process` *Interactions from KGML*

Description

Processes KEGG Pathways data extracted from a KGML file. Joins the entries and relations into a single data frame and translates the Gene Symbols to UniProt IDs.

Usage

```
kegg_process(entries, relations, max_expansion = NULL, simplify = FALSE)
```

Arguments

`entries`	A data frames with entries extracted from a KGML file by `kegg_pathway_download`.
`relations`	A data frames with relations extracted from a KGML file by `kegg_pathway_download`.
`max_expansion`	Numeric: the maximum number of relations derived from a single relation record. As one entry might represent more than one molecular entities, one relation might yield a large number of relations in the processing. This happens in a combinatorial way, e.g. if the two entries represent 3 and 4 entities, that results 12 relations. If `NULL`, all relations will be expanded.
`simplify`	Logical: remove KEGG's internal identifiers and the pathway annotations, keep only unique interactions with direction and effect sign.

Value

A data frame (tibble) of interactions. In rare cases when a pathway doesn't contain any relation, returns `NULL`.

See Also

- `kegg_pathway_download`
- `kegg_pathways_download`
- `kegg_pathway_list`

Examples

```
hsa04350 <- kegg_pathway_download('hsa04350', process = FALSE)
tgf_pathway <- kegg_process(hsa04350$entries, hsa04350$relations)
tgf_pathway
# # A tibble: 50 x 12
#    source target type  effect arrow relation_id kegg_id_source
#    <chr>  <chr>  <chr> <chr>  <chr> <chr>       <chr>
#  1 51     49     PPrel activ. -->   hsa04350:1  hsa:7040 hsa:.
#  2 57     55     PPrel activ. -->   hsa04350:2  hsa:151449 hs.
#  3 34     32     PPrel activ. -->   hsa04350:3  hsa:3624 hsa:.
#  4 20     17     PPrel activ. -->   hsa04350:4  hsa:4838
#  5 60     46     PPrel activ. -->   hsa04350:5  hsa:4086 hsa:.
# # . with 45 more rows, and 5 more variables: genesymbol_source <chr>,
# #   uniprot_source <chr>, kegg_id_target <chr>,
# #   genesymbol_target <chr>, uniprot_target <chr>
```

`latin_name` *Latin (scientific) names of organisms*

Description

Latin (scientific) names of organisms

Usage

```
latin_name(name)
```

Arguments

`name`	Vector with any kind of organism name or identifier, can be also mixed type.

Value

Character vector with latin (scientific) names, NA if a name in the input could not be found.

See Also

- `ncbi_taxid`
- `common_name`
- `ensembl_name`

Examples

```
latin_name(c(9606, "cat", "dog"))
# [1] "Homo sapiens" "Felis catus" "Canis lupus familiaris"
latin_name(c(9606, "cat", "doggy"))
# [1] "Homo sapiens" "Felis catus"  NA
```

`load_db` *Load a built in database*

Description

Load a built in database

Usage

```
load_db(key, param = list())
```

Arguments

`key`	Character: the key of the database to load. For a list of available keys see `omnipath_show_db`.
`param`	List: override the defaults or pass further parameters to the database loader function. See the loader functions and their default parameters in `omnipath_show_db`.

Details

This function loads a database which is stored within the package namespace until its expiry. The loaded database is accessible by `get_db` and the loading process is typically initiated by `get_db`, not by the users directly.

Value

Returns `NULL`.

See Also

`omnipath_show_db`, `get_db`

Examples

```
load_db('go_slim')
omnipath_show_db()
```

`ncbi_taxid` *NCBI Taxonomy IDs of organisms*

Description

NCBI Taxonomy IDs of organisms

Usage

```
ncbi_taxid(name)
```

Arguments

`name`	Vector with any kind of organism name or identifier, can be also mixed type.

Value

Integer vector with NCBI Taxonomy IDs, NA if a name in the input could not be found.

See Also

- `latin_name`
- `common_name`
- `ensembl_name`

Examples

```
ncbi_taxid(c("Homo sapiens", "cat", "dog"))
# [1] 9606 9685 9615
ncbi_taxid(c(9606, "cat", "doggy"))
# [1] 9606 9685   NA
```

`nichenet_build_model` *Construct a NicheNet ligand-target model*

Description

Construct a NicheNet ligand-target model

Usage

```
nichenet_build_model(optimization_results, networks, use_weights = TRUE)
```

Arguments

`optimization_results`	The outcome of NicheNet parameter optimization as produced by `nichenet_optimization`.
`networks`	A list with NicheNet format signaling, ligand-receptor and gene regulatory networks as produced by `nichenet_networks`.
`use_weights`	Logical: whether to use the optimized weights.

Value

A named list with two elements: ‘weighted_networks‘ and ‘optimized_parameters‘.

Examples

```
## Not run:
expression <- nichenet_expression_data()
networks <- nichenet_networks()
optimization_results <- nichenet_optimization(networks, expression)
nichenet_model <- nichenet_build_model(optimization_results, networks)

## End(Not run)
```

`nichenet_expression_data` *Expression data from ligand-receptor perturbation experiments used by NicheNet*

Description

NicheNet uses expression data from a collection of published ligand or receptor KO or perturbation experiments to build its model. This function retrieves the original expression data, deposited in Zenodo (`https://zenodo.org/record/3260758`).

Usage

```
nichenet_expression_data()
```

Value

Nested list, each element contains a data frame of processed expression data and key variables about the experiment.

Examples

```
exp_data <- nichenet_expression_data()
head(names(exp_data))
# [1] "bmp4_tgfb"     "tgfb_bmp4"     "nodal_Nodal"   "spectrum_Il4"
# [5] "spectrum_Tnf"  "spectrum_Ifng"
purrr::map_chr(head(exp_data), 'from')
#     bmp4_tgfb     tgfb_bmp4   nodal_Nodal  spectrum_Il4  spectrum_Tnf
#       "BMP4"       "TGFB1"       "NODAL"         "IL4"         "TNF"
# spectrum_Ifng
#       "IFNG"
```

`nichenet_gr_network` *Builds a NicheNet gene regulatory network*

Description

Builds gene regulatory network prior knowledge for NicheNet using multiple resources.

Usage

```
nichenet_gr_network(
  omnipath = list(),
  harmonizome = list(),
  regnetwork = list(),
  htridb = list(),
  remap = list(),
  evex = list(),
  pathwaycommons = list(),
  trrust = list(),
  only_omnipath = FALSE
)
```

Arguments

`omnipath`	List with paramaters to be passed to `nichenet_gr_network_omnipath`.
`harmonizome`	List with paramaters to be passed to `nichenet_gr_network_harmonizome`.
`regnetwork`	List with paramaters to be passed to `nichenet_gr_network_regnetwork`.
`htridb`	List with paramaters to be passed to `nichenet_gr_network_htridb`.
`remap`	List with paramaters to be passed to `nichenet_gr_network_remap`.
`evex`	List with paramaters to be passed to `nichenet_gr_network_evex`.
`pathwaycommons`	List with paramaters to be passed to `nichenet_gr_network_pathwaycommons`.
`trrust`	List with paramaters to be passed to `nichenet_gr_network_trrust`.
`only_omnipath`	Logical: a shortcut to use only OmniPath as network resource.

Value

A network data frame (tibble) with gene regulatory interactions suitable for use with NicheNet.

See Also

- `nichenet_gr_network_evex`
- `nichenet_gr_network_harmonizome`
- `nichenet_gr_network_htridb`
- `nichenet_gr_network_omnipath`
- `nichenet_gr_network_pathwaycommons`
- `nichenet_gr_network_regnetwork`
- `nichenet_gr_network_remap`
- `nichenet_gr_network_trrust`

Examples

```
# load everything with the default parameters:
gr_network <- nichenet_gr_network()

# less targets from ReMap, not using RegNetwork:
gr_network <- nichenet_gr_network(
    # I needed to disable ReMap here due to some issues
    # of one of the Bioconductor build servers
    # remap = list(top_targets = 200),
    remap = NULL,
    regnetwork = NULL,
)

# use only OmniPath:
gr_network_omnipath <- nichenet_gr_network(only_omnipath = TRUE)
```

`nichenet_gr_network_evex`

NicheNet gene regulatory network from EVEX

Description

Builds a gene regulatory network using data from the EVEX database and converts it to a format suitable for NicheNet.

Usage

```
nichenet_gr_network_evex(
  top_confidence = 0.75,
  indirect = FALSE,
  regulation_of_expression = FALSE
)
```

Arguments

`top_confidence`	Double, between 0 and 1. Threshold based on the quantile of the confidence score.
`indirect`	Logical: whether to include indirect interactions.
`regulation_of_expression`	Logical: whether to include also the "regulation of expression" type interactions.

Value

Data frame of interactions in NicheNet format.

Data frame with gene regulatory interactions in NicheNet format.

See Also

- `nichenet_gr_network`
- `evex_download`

Examples

```
# use only the 10% with the highest confidence:
evex_gr_network <- nichenet_gr_network_evex(top_confidence = .9)
```

`nichenet_gr_network_harmonizome` *NicheNet gene regulatory network from Harmonizome*

Description

Builds gene regulatory network prior knowledge for NicheNet using Harmonizome

Usage

```
nichenet_gr_network_harmonizome(
  datasets = c("cheappi", "encodetfppi", "jasparpwm", "transfac", "transfacpwm",
    "motifmap", "geotf", "geokinase", "geogene"),
  ...
)
```

Arguments

`datasets`	The datasets to use. For possible values please refer to default value and the Harmonizome webpage.
`...`	Ignored.

Value

Data frame with gene regulatory interactions in NicheNet format.

See Also

- `nichenet_gr_network`
- `harmonizome_download`

Examples

```
# use only JASPAR and TRANSFAC:
hz_gr_network <- nichenet_gr_network_harmonizome(
    datasets = c('jasparpwm', 'transfac', 'transfacpwm')
)
```

`nichenet_gr_network_htridb`

NicheNet gene regulatory network from HTRIdb

Description

Builds a gene regulatory network using data from the HTRIdb database and converts it to a format suitable for NicheNet.

Usage

```
nichenet_gr_network_htridb()
```

Value

Data frame with gene regulatory interactions in NicheNet format.

See Also

`htridb_download`, `nichenet_gr_network`

Examples

```
htri_gr_network <- nichenet_gr_network_htridb()
```

`nichenet_gr_network_omnipath`

Builds gene regulatory network for NicheNet using OmniPath

Description

Retrieves network prior knowledge from OmniPath and provides it in a format suitable for NicheNet. This method never downloads the ‘ligrecextra‘ dataset because the ligand-receptor interactions are supposed to come from `nichenet_lr_network_omnipath`.

Usage

```
nichenet_gr_network_omnipath(min_curation_effort = 0, ...)
```

Arguments

`min_curation_effort`	Lower threshold for curation effort
`...`	Passed to `import_transcriptional_interactions`

Value

A network data frame (tibble) with gene regulatory interactions suitable for use with NicheNet.

See Also

- `nichenet_gr_network_evex`
- `nichenet_gr_network_harmonizome`
- `nichenet_gr_network_htridb`
- `nichenet_gr_network_omnipath`
- `nichenet_gr_network_pathwaycommons`
- `nichenet_gr_network_regnetwork`
- `nichenet_gr_network_remap`
- `nichenet_gr_network_trrust`

Examples

```
# use interactions up to confidence level "C" from DoRothEA:
op_gr_network <- nichenet_gr_network_omnipath(
    dorothea_levels = c('A', 'B', 'C')
)
```

`nichenet_gr_network_pathwaycommons` *NicheNet gene regulatory network from PathwayCommons*

Description

Builds gene regulation prior knowledge for NicheNet using PathwayCommons.

Usage

```
nichenet_gr_network_pathwaycommons(
  interaction_types = "controls-expression-of",
  ...
)
```

Arguments

`interaction_types`	Character vector with PathwayCommons interaction types. Please refer to the default value and the PathwayCommons webpage.
`...`	Ignored.

Value

Data frame with gene regulatory interactions in NicheNet format.

See Also

- `nichenet_gr_network`
- `pathwaycommons_download`

Examples

```
pc_gr_network <- nichenet_gr_network_pathwaycommons()
```

`nichenet_gr_network_regnetwork`

NicheNet gene regulatory network from RegNetwork

Description

Builds a gene regulatory network using data from the RegNetwork database and converts it to a format suitable for NicheNet.

Usage

```
nichenet_gr_network_regnetwork()
```

Value

Data frame with gene regulatory interactions in NicheNet format.

See Also

- `regnetwork_download`
- `nichenet_gr_network`

Examples

```
regn_gr_network <- nichenet_gr_network_regnetwork()
```

`nichenet_gr_network_remap`
NicheNet gene regulatory network from ReMap

Description

Builds a gene regulatory network using data from the ReMap database and converts it to a format suitable for NicheNet.

Usage

```
nichenet_gr_network_remap(
  score = 100,
  top_targets = 500,
  only_known_tfs = TRUE
)
```

Arguments

`score`	Numeric: a minimum score between 0 and 1000, records with lower scores will be excluded. If NULL no filtering performed.
`top_targets`	Numeric: the number of top scoring targets for each TF. Essentially the maximum number of targets per TF. If NULL the number of targets is not restricted.
`only_known_tfs`	Logical: whether to exclude TFs which are not in TF census.

Value

Data frame with gene regulatory interactions in NicheNet format.

See Also

- `remap_filtered`
- `nichenet_gr_network`

Examples

```
# use only max. top 100 targets for each TF:
remap_gr_network <- nichenet_gr_network_remap(top_targets = 100)
```

`nichenet_gr_network_trrust`

NicheNet gene regulatory network from TRRUST

Description

Builds a gene regulatory network using data from the TRRUST database and converts it to a format suitable for NicheNet.

Usage

```
nichenet_gr_network_trrust()
```

Value

Data frame with gene regulatory interactions in NicheNet format.

See Also

- `trrust_download`
- `nichenet_gr_network`

Examples

```
trrust_gr_network <- nichenet_gr_network_trrust()
```

`nichenet_ligand_activities`

Calls the NicheNet ligand activity analysis

Description

Calls the NicheNet ligand activity analysis

Usage

```
nichenet_ligand_activities(
  ligand_target_matrix,
  lr_network,
  expressed_genes_transmitter,
  expressed_genes_receiver,
  genes_of_interest,
  background_genes = NULL,
  n_top_ligands = 42,
  n_top_targets = 250
)
```

Arguments

`ligand_target_matrix`	A matrix with rows and columns corresponding to ligands and targets, respectively. Produced by `nichenet_ligand_target_matrix` or `nichenetr::construct_ligand_targe`
`lr_network`	A data frame with ligand-receptor interactions, as produced by `nichenet_lr_network`.
`expressed_genes_transmitter`	Character vector with the gene symbols of the genes expressed in the cells transmitting the signal.
`expressed_genes_receiver`	Character vector with the gene symbols of the genes expressed in the cells receiving the signal.
`genes_of_interest`	Character vector with the gene symbols of the genes of interest. These are the genes in the receiver cell population that are potentially affected by ligands expressed by interacting cells (e.g. genes differentially expressed upon cell-cell interaction).
`background_genes`	Character vector with the gene symbols of the genes to be used as background.
`n_top_ligands`	How many of the top ligands to include in the ligand-target table.
`n_top_targets`	For each ligand, how many of the top targets to include in the ligand-target table.

Value

A named list with ‘ligand_activities‘ (a tibble giving several ligand activity scores; following columns in the tibble: $test_ligand, $auroc, $aupr and $pearson) and ‘ligand_target_links‘ (a tibble with columns ligand, target and weight (i.e. regulatory potential score)).

Examples

```
## Not run:
networks <- nichenet_networks()
expression <- nichenet_expression_data()
optimization_results <- nichenet_optimization(networks, expression)
nichenet_model <- nichenet_build_model(optimization_results, networks)
lt_matrix <- nichenet_ligand_target_matrix(
    nichenet_model$weighted_networks,
    networks$lr_network,
    nichenet_model$optimized_parameters
)
ligand_activities <- nichenet_ligand_activities(
    ligand_target_matrix = lt_matrix,
    lr_network = networks$lr_network,
    # the rest of the parameters should come
    # from your transcriptomics data:
    expressed_genes_transmitter = expressed_genes_transmitter,
    expressed_genes_receiver = expressed_genes_receiver,
    genes_of_interest = genes_of_interest
)

## End(Not run)
```

`nichenet_ligand_target_links`

Compiles a table with weighted ligand-target links

Description

A wrapper around `nichenetr::get_weighted_ligand_target_links` to compile a data frame with weighted links from the top ligands to their top targets.

Usage

```
nichenet_ligand_target_links(
  ligand_activities,
  ligand_target_matrix,
  genes_of_interest,
  n_top_ligands = 42,
  n_top_targets = 250
)
```

Arguments

`ligand_activities`	Ligand activity table as produced by `nichenetr::predict_ligand_activities`.
`ligand_target_matrix`	Ligand-target matrix as produced by `nichenetr::construct_ligand_target_matrix` or the wrapper around it in the current package: `nichenet_ligand_target_matrix`.
`genes_of_interest`	Character vector with the gene symbols of the genes of interest. These are the genes in the receiver cell population that are potentially affected by ligands expressed by interacting cells (e.g. genes differentially expressed upon cell-cell interaction).
`n_top_ligands`	How many of the top ligands to include in the ligand-target table.
`n_top_targets`	For each ligand, how many of the top targets to include in the ligand-target table.

Value

A tibble with columns ligand, target and weight (i.e. regulatory potential score).

Examples

```
## Not run:
networks <- nichenet_networks()
expression <- nichenet_expression_data()
optimization_results <- nichenet_optimization(networks, expression)
nichenet_model <- nichenet_build_model(optimization_results, networks)
lt_matrix <- nichenet_ligand_target_matrix(
    nichenet_model$weighted_networks,
    networks$lr_network,
    nichenet_model$optimized_parameters
)
ligand_activities <- nichenet_ligand_activities(
```

```
        ligand_target_matrix = lt_matrix,
        lr_network = networks$lr_network,
        # the rest of the parameters should come
        # from your transcriptomics data:
        expressed_genes_transmitter = expressed_genes_transmitter,
        expressed_genes_receiver = expressed_genes_receiver,
        genes_of_interest = genes_of_interest
    )
    lt_links <- nichenet_ligand_target_links(
        ligand_activities = ligand_activities,
        ligand_target_matrix = lt_matrix,
        genes_of_interest = genes_of_interest,
        n_top_ligands = 20,
        n_top_targets = 100
    )

    ## End(Not run)
```

`nichenet_ligand_target_matrix` *Creates a NicheNet ligand-target matrix*

Description

Creates a NicheNet ligand-target matrix

Usage

```
nichenet_ligand_target_matrix(
  weighted_networks,
  lr_network,
  optimized_parameters,
  use_weights = TRUE,
  construct_ligand_target_matrix_param = list()
)
```

Arguments

`weighted_networks`	Weighted networks as provided by `nichenet_build_model`.
`lr_network`	A data frame with ligand-receptor interactions, as produced by `nichenet_lr_network`.
`optimized_parameters`	The outcome of NicheNet parameter optimization as produced by `nichenet_build_model`.
`use_weights`	Logical: wether the network sources are weighted. In this function it only affects the output file name.
`construct_ligand_target_matrix_param`	Override parameters for `nichenetr::construct_ligand_target_matrix`.

Value

A matrix containing ligand-target probability scores.

Examples

```
## Not run:
networks <- nichenet_networks()
expression <- nichenet_expression_data()
optimization_results <- nichenet_optimization(networks, expression)
nichenet_model <- nichenet_build_model(optimization_results, networks)
lt_matrix <- nichenet_ligand_target_matrix(
    nichenet_model$weighted_networks,
    networks$lr_network,
    nichenet_model$optimized_parameters
)

## End(Not run)
```

`nichenet_lr_network` *Builds a NicheNet ligand-receptor network*

Description

Builds ligand-receptor network prior knowledge for NicheNet using multiple resources.

Usage

```
nichenet_lr_network(
  omnipath = list(),
  guide2pharma = list(),
  ramilowski = list(),
  only_omnipath = FALSE,
  quality_filter_param = list()
)
```

Arguments

`omnipath`	List with paramaters to be passed to `nichenet_lr_network_omnipath`.
`guide2pharma`	List with paramaters to be passed to `nichenet_lr_network_guide2pharma`.
`ramilowski`	List with paramaters to be passed to `nichenet_lr_network_ramilowski`.
`only_omnipath`	Logical: a shortcut to use only OmniPath as network resource.
`quality_filter_param`	Arguments for `filter_intercell_network` (quality filtering of the OmniPath ligand-receptor network). It is recommended to check these parameters and apply some quality filtering. The defaults already ensure certain filtering, but you might want more relaxed or stringent options.

Value

A network data frame (tibble) with ligand-receptor interactions suitable for use with NicheNet.

See Also

- nichenet_lr_network_omnipath
- nichenet_lr_network_guide2pharma
- nichenet_lr_network_ramilowski
- filter_intercell_network

Examples

```
# load everything with the default parameters:
lr_network <- nichenet_lr_network()

# don't use Ramilowski:
lr_network <- nichenet_lr_network(ramilowski = NULL)

# use only OmniPath:
lr_network_omnipath <- nichenet_lr_network(only_omnipath = TRUE)
```

`nichenet_lr_network_guide2pharma`

Ligand-receptor network from Guide to Pharmacology

Description

Downloads ligand-receptor interactions from the Guide to Pharmacology database and converts it to a format suitable for NicheNet.

Usage

```
nichenet_lr_network_guide2pharma()
```

Value

Data frame with ligand-receptor interactions in NicheNet format.

See Also

nichenet_lr_network, guide2pharma_download

Examples

```
g2p_lr_network <- nichenet_lr_network_guide2pharma()
```

`nichenet_lr_network_omnipath` *Builds ligand-receptor network for NicheNet using OmniPath*

Description

Retrieves network prior knowledge from OmniPath and provides it in a format suitable for NicheNet. This method never downloads the ‘ligrecextra‘ dataset because the ligand-receptor interactions are supposed to come from `nichenet_lr_network_omnipath`.

Usage

```
nichenet_lr_network_omnipath(quality_filter_param = list(), ...)
```

Arguments

`quality_filter_param`	List with arguments for `filter_intercell_network`. It is recommended to check these parameters and apply some quality filtering. The defaults already ensure certain filtering, but you might want more relaxed or stringent options.
`...`	Passed to `import_intercell_network`

Value

A network data frame (tibble) with ligand-receptor interactions suitable for use with NicheNet.

See Also

- `nichenet_lr_network`
- `import_intercell_network`

Examples

```
# use only ligand-receptor interactions (not for example ECM-adhesion):
op_lr_network <- nichenet_lr_network_omnipath(ligand_receptor = TRUE)

# use only CellPhoneDB and Guide to Pharmacology:
op_lr_network <- nichenet_lr_network_omnipath(
    resources = c('CellPhoneDB', 'Guide2Pharma')
)

# only interactions where the receiver is a transporter:
op_lr_network <- nichenet_lr_network_omnipath(
    receiver_param = list(parent = 'transporter')
)
```

`nichenet_lr_network_ramilowski` *Ligand-receptor network from Ramilowski 2015*

Description

Downloads ligand-receptor interactions from Supplementary Table 2 of the paper 'A draft network of ligand–receptor-mediated multicellular signalling in human' (Ramilowski et al. 2015, `https://www.nature.com/articles/ncomms8866`). It converts the downloaded table to a format suitable for NicheNet.

Usage

```
nichenet_lr_network_ramilowski(
  evidences = c("literature supported", "putative")
)
```

Arguments

`evidences`	Character: evidence types, "literature supported", "putative" or both.

Value

Data frame with ligand-receptor interactions in NicheNet format.

See Also

- `nichenet_lr_network`
- `ramilowski_download`

Examples

```
# use only the literature supported data:
rami_lr_network <- nichenet_lr_network_ramilowski(
    evidences = 'literature supported'
)
```

`nichenet_main` *Executes the full NicheNet pipeline*

Description

Builds all prior knowledge data required by NicheNet. For this it calls a multitude of methods to download and combine data from various databases according to the settings. The content of the prior knowledge data is highly customizable, see the documentation of the related functions. After the prior knowledge is ready, it performs parameter optimization to build a NicheNet model. This results a weighted ligand- target matrix. Then, considering the expressed genes from user provided data, a gene set of interest and background genes, it executes the NicheNet ligand activity analysis.

Usage

```
nichenet_main(
  only_omnipath = FALSE,
  expressed_genes_transmitter = NULL,
  expressed_genes_receiver = NULL,
  genes_of_interest = NULL,
  background_genes = NULL,
  use_weights = TRUE,
  n_top_ligands = 42,
  n_top_targets = 250,
  signaling_network = list(),
  lr_network = list(),
  gr_network = list(),
  small = FALSE,
  tiny = FALSE,
  make_multi_objective_function_param = list(),
  objective_function_param = list(),
  mlrmbo_optimization_param = list(),
  construct_ligand_target_matrix_param = list(),
  results_dir = NULL,
  quality_filter_param = list()
)
```

Arguments

`only_omnipath`	Logical: use only OmniPath for network knowledge. This is a simple switch for convenience, further options are available by the other arguments. By default we use all available resources. The networks can be customized on a resource by resource basis, as well as providing custom parameters for individual resources, using the parameters ‘signaling_network‘, ‘lr_network‘ and ‘gr_network‘.
`expressed_genes_transmitter`	Character vector with the gene symbols of the genes expressed in the cells transmitting the signal.
`expressed_genes_receiver`	Character vector with the gene symbols of the genes expressed in the cells receiving the signal.
`genes_of_interest`	Character vector with the gene symbols of the genes of interest. These are the genes in the receiver cell population that are potentially affected by ligands expressed by interacting cells (e.g. genes differentially expressed upon cell-cell interaction).
`background_genes`	Character vector with the gene symbols of the genes to be used as background.
`use_weights`	Logical: calculate and use optimized weights for resources (i.e. one resource seems to be better than another, hence the former is considered with a higher weight).
`n_top_ligands`	How many of the top ligands to include in the ligand-target table.
`n_top_targets`	How many of the top targets (for each of the top ligands) to consider in the ligand-target table.
`signaling_network`	A list of parameters for building the signaling network, passed to `nichenet_signaling_network`.

`lr_network`	A list of parameters for building the ligand-receptor network, passed to `nichenet_lr_network`.
`gr_network`	A list of parameters for building the gene regulatory network, passed to `nichenet_gr_network`.
`small`	Logical: build a small network for testing purposes, using only OmniPath data. It is also a high quality network, it is reasonable to try the analysis with this small network.
`tiny`	Logical: build an even smaller network for testing purposes. As this involves random subsetting, it's not recommended to use this network for analysis.
`make_multi_objective_function_param`	Override parameters for `smoof::makeMultiObjectiveFunction`.
`objective_function_param`	Override additional arguments passed to the objective function.
`mlrmbo_optimization_param`	Override arguments for `nichenetr::mlrmbo_optimization`.
`construct_ligand_target_matrix_param`	Override parameters for `nichenetr::construct_ligand_target_matrix`.
`results_dir`	Character: path to the directory to save intermediate and final outputs from NicheNet methods.
`quality_filter_param`	Arguments for `filter_intercell_network` (quality filtering of the OmniPath ligand-receptor network). It is recommended to check these parameters and apply some quality filtering. The defaults already ensure certain filtering, but you might want more relaxed or stringent options.

Details

About *small* and *tiny* networks: Building a NicheNet model is computationally demanding, taking several hours to run. As this is related to the enormous size of the networks, to speed up testing we can use smaller networks, around 1,000 times smaller, with few thousands of interactions instead of few millions. Random subsetting of the whole network would result disjunct fragments, instead we load only a few resources. To run the whole pipeline with tiny networks use `nichenet_test`.

Value

A named list with the intermediate and final outputs of the pipeline: ‘networks‘, ‘expression‘, ‘optimized_parameters‘, ‘weighted_networks‘ and ‘ligand_target_matrix‘.

See Also

- `nichenet_networks`
- `nichenet_signaling_network`
- `nichenet_lr_network`
- `nichenet_gr_network`
- `nichenet_test`
- `nichenet_workarounds`
- `nichenet_results_dir`

Examples

```
## Not run:
nichenet_results <- nichenet_main(
    # altering some network resource parameters, the rest
    # of the resources will be loaded according to the defaults
    signaling_network = list(
        cpdb = NULL, # this resource will be excluded
        inbiomap = NULL,
        evex = list(min_confidence = 1.0) # override some parameters
    ),
    gr_network = list(only_omnipath = TRUE),
    n_top_ligands = 20,
    # override the default number of CPU cores to use
    mlrmbo_optimization_param = list(ncores = 4)
)

## End(Not run)
```

`nichenet_networks` *Builds NicheNet network prior knowledge*

Description

Builds network knowledge required by NicheNet. For this it calls a multitude of methods to download and combine data from various databases according to the settings. The content of the prior knowledge data is highly customizable, see the documentation of the related functions.

Usage

```
nichenet_networks(
  signaling_network = list(),
  lr_network = list(),
  gr_network = list(),
  only_omnipath = FALSE,
  small = FALSE,
  tiny = FALSE,
  quality_filter_param = list()
)
```

Arguments

`signaling_network`	A list of parameters for building the signaling network, passed to `nichenet_signaling_network`
`lr_network`	A list of parameters for building the ligand-receptor network, passed to `nichenet_lr_network`
`gr_network`	A list of parameters for building the gene regulatory network, passed to `nichenet_gr_network`
`only_omnipath`	Logical: a shortcut to use only OmniPath as network resource.
`small`	Logical: build a small network for testing purposes, using only OmniPath data. It is also a high quality network, it is reasonable to try the analysis with this small network.

`tiny`	Logical: build an even smaller network for testing purposes. As this involves random subsetting, it's not recommended to use this network for analysis.
`quality_filter_param`	Arguments for `filter_intercell_network` (quality filtering of the OmniPath ligand-receptor network). It is recommended to check these parameters and apply some quality filtering. The defaults already ensure certain filtering, but you might want more relaxed or stringent options.

Value

A named list with three network data frames (tibbles): the signaling, the ligand-receptor (lr) and the gene regulatory (gr) networks.

See Also

- `nichenet_signaling_network`
- `nichenet_lr_network`
- `nichenet_gr_network`

Examples

```
## Not run:
networks <- nichenet_networks()
dplyr::sample_n(networks$gr_network, 10)
# # A tibble: 10 x 4
#    from    to       source               database
#    <chr>   <chr>    <chr>                <chr>
#  1 MAX     ALG3     harmonizome_ENCODE   harmonizome
#  2 MAX     IMPDH1   harmonizome_ENCODE   harmonizome
#  3 SMAD5   LCP1     Remap_5              Remap
#  4 HNF4A   TNFRSF19 harmonizome_CHEA     harmonizome
#  5 SMC3    FAP      harmonizome_ENCODE   harmonizome
#  6 E2F6    HIST1H1B harmonizome_ENCODE   harmonizome
#  7 TFAP2C  MAT2B    harmonizome_ENCODE   harmonizome
#  8 USF1    TBX4     harmonizome_TRANSFAC harmonizome
#  9 MIR133B FETUB    harmonizome_TRANSFAC harmonizome
# 10 SP4     HNRNPH2  harmonizome_ENCODE   harmonizome

## End(Not run)

# use only OmniPath:
omnipath_networks <- nichenet_networks(only_omnipath = TRUE)
```

`nichenet_optimization` *Optimizes NicheNet model parameters*

Description

Optimize NicheNet method parameters, i.e. PageRank parameters and source weights, basedon a collection of experiments where the effect of a ligand on gene expression was measured.

Usage

```
nichenet_optimization(
  networks,
  expression,
  make_multi_objective_function_param = list(),
  objective_function_param = list(),
  mlrmbo_optimization_param = list()
)
```

Arguments

`networks`	A list with NicheNet format signaling, ligand-receptor and gene regulatory networks as produced by `nichenet_networks`.
`expression`	A list with expression data from ligand perturbation experiments, as produced by `nichenet_expression_data`.
`make_multi_objective_function_param`	Override parameters for `smoof::makeMultiObjectiveFunction`.
`objective_function_param`	Override additional arguments passed to the objective function.
`mlrmbo_optimization_param`	Override arguments for `nichenetr::mlrmbo_optimization`.

Value

A result object from the function `mlrMBO::mbo`. Among other things, this contains the optimal parameter settings, the output corresponding to every input etc.

Examples

```
## Not run:
networks <- nichenet_networks()
expression <- nichenet_expression_data()
optimization_results <- nichenet_optimization(networks, expression)

## End(Not run)
```

`nichenet_remove_orphan_ligands`

Removes experiments with orphan ligands

Description

Removes from the expression data the perturbation experiments involving ligands without connections.

Usage

```
nichenet_remove_orphan_ligands(expression, lr_network)
```

Arguments

`expression`	Expression data as returned by `nichenet_expression_data`.
`lr_network`	A NicheNet format ligand-recptor network data frame as produced by `nichenet_lr_network`.

Value

The same list as ‘expression‘ with certain elements removed.

Examples

```
lr_network <- nichenet_lr_network()
expression <- nichenet_expression_data()
expression <- nichenet_remove_orphan_ligands(expression, lr_network)
```

`nichenet_results_dir` *Path to the current NicheNet results directory*

Description

Path to the directory to save intermediate and final outputs from NicheNet methods.

Usage

```
nichenet_results_dir()
```

Value

Character: path to the NicheNet results directory.

Examples

```
nichenet_results_dir()
# [1] "nichenet_results"
```

`nichenet_signaling_network` *Builds a NicheNet signaling network*

Description

Builds signaling network prior knowledge for NicheNet using multiple resources.

Usage

```
nichenet_signaling_network(
  omnipath = list(),
  pathwaycommons = list(),
  harmonizome = list(),
  vinayagam = list(),
  cpdb = list(),
  evex = list(),
  inbiomap = list(),
  only_omnipath = FALSE
)
```

Arguments

`omnipath`	List with paramaters to be passed to `nichenet_signaling_network_omnipath`.
`pathwaycommons`	List with paramaters to be passed to `nichenet_signaling_network_pathwaycommons`.
`harmonizome`	List with paramaters to be passed to `nichenet_signaling_network_harmonizome`.
`vinayagam`	List with paramaters to be passed to `nichenet_signaling_network_vinayagam`.
`cpdb`	List with paramaters to be passed to `nichenet_signaling_network_cpdb`.
`evex`	List with paramaters to be passed to `nichenet_signaling_network_evex`.
`inbiomap`	List with paramaters to be passed to `nichenet_signaling_network_inbiomap`.
`only_omnipath`	Logical: a shortcut to use only OmniPath as network resource.

Value

A network data frame (tibble) with signaling interactions suitable for use with NicheNet.

See Also

- `nichenet_signaling_network_omnipath`
- `nichenet_signaling_network_pathwaycommons`
- `nichenet_signaling_network_harmonizome`
- `nichenet_signaling_network_vinayagam`
- `nichenet_signaling_network_cpdb`
- `nichenet_signaling_network_evex`
- `nichenet_signaling_network_inbiomap`

Examples

```
# load everything with the default parameters:
# we don't load inBio Map due to the - hopefully
# temporary - issues of their server
sig_network <- nichenet_signaling_network(inbiomap = NULL, cpdb = NULL)

# override parameters for some resources:
sig_network <- nichenet_signaling_network(
    omnipath = list(resources = c('SIGNOR', 'SignaLink3', 'SPIKE')),
    pathwaycommons = NULL,
    harmonizome = list(datasets = c('phosphositeplus', 'depod')),
    # we can not include this in everyday tests as it takes too long:
```

```
    # cpdb = list(complex_max_size = 1, min_score = .98),
    cpdb = NULL,
    evex = list(min_confidence = 1.5),
    inbiomap = NULL
)

# use only OmniPath:
sig_network_omnipath <- nichenet_signaling_network(only_omnipath = TRUE)
```

---

nichenet_signaling_network_cpdb

Builds signaling network for NicheNet using ConsensusPathDB

Description

Builds signaling network prior knowledge using ConsensusPathDB (CPDB) data. Note, the interactions from CPDB are not directed and many of them comes from complex expansion. Find out more at `http://cpdb.molgen.mpg.de/`.

Usage

```
nichenet_signaling_network_cpdb(...)
```

Arguments

`...`	Passed to `consensuspathdb_download`.

Value

A network data frame (tibble) with signaling interactions suitable for use with NicheNet.

See Also

- `nichenet_signaling_network`
- `consensuspathdb_download`

Examples

```
# use some parameters stricter than default:
cpdb_signaling_network <- nichenet_signaling_network_cpdb(
    complex_max_size = 2,
    min_score = .99
)
```

`nichenet_signaling_network_evex` *NicheNet signaling network from EVEX*

Description

Builds signaling network prior knowledge for NicheNet from the EVEX database.

Usage

```
nichenet_signaling_network_evex(top_confidence = 0.75, indirect = FALSE, ...)
```

Arguments

`top_confidence`	Double, between 0 and 1. Threshold based on the quantile of the confidence score.
`indirect`	Logical: whether to include indirect interactions.
`...`	Ignored.

Value

A network data frame (tibble) with signaling interactions suitable for use with NicheNet.

See Also

- `evex_download`
- `nichenet_signaling_network`

Examples

```
ev_signaling_network <- nichenet_signaling_network_evex(
    top_confidence = .9
)
```

`nichenet_signaling_network_harmonizome` *NicheNet signaling network from Harmonizome*

Description

Builds signaling network prior knowledge for NicheNet using Harmonizome

Usage

```
nichenet_signaling_network_harmonizome(
  datasets = c("phosphositeplus", "kea", "depod"),
  ...
)
```

Arguments

`datasets`	The datasets to use. For possible values please refer to default value and the Harmonizome webpage.
`...`	Ignored.

Value

A network data frame (tibble) with signaling interactions suitable for use with NicheNet.

Examples

```
# use only KEA and PhosphoSite:
hz_signaling_network <- nichenet_signaling_network_harmonizome(
    datasets = c('kea', 'phosphositeplus')
)
```

`nichenet_signaling_network_inbiomap`

NicheNet signaling network from InWeb InBioMap

Description

Builds signaling network prior knowledge for NicheNet from the InWeb InBioMap database.

Usage

```
nichenet_signaling_network_inbiomap(...)
```

Arguments

`...`	Ignored.

Value

A network data frame (tibble) with signaling interactions suitable for use with NicheNet.

See Also

`nichenet_signaling_network`, `inbiomap_download`

Examples

```
## Not run:
ib_signaling_network <- nichenet_signaling_network_inbiomap()

## End(Not run)
```

`nichenet_signaling_network_omnipath` *Builds signaling network for NicheNet using OmniPath*

Description

Retrieves network prior knowledge from OmniPath and provides it in a format suitable for NicheNet. This method never downloads the ‘ligrecextra‘ dataset because the ligand-receptor interactions are supposed to come from `nichenet_lr_network_omnipath`.

Usage

```
nichenet_signaling_network_omnipath(min_curation_effort = 0, ...)
```

Arguments

`min_curation_effort`	Lower threshold for curation effort
`...`	Passed to `import_post_translational_interactions`

Value

A network data frame (tibble) with signaling interactions suitable for use with NicheNet.

See Also

- `nichenet_signaling_network`

Examples

```
# use interactions with at least 2 evidences (reference or database)
op_signaling_network <- nichenet_signaling_network_omnipath(
    min_curation_effort = 2
)
```

`nichenet_signaling_network_pathwaycommons` *NicheNet signaling network from PathwayCommons*

Description

Builds signaling network prior knowledge for NicheNet using PathwayCommons.

Usage

```
nichenet_signaling_network_pathwaycommons(
  interaction_types = c("catalysis-precedes", "controls-phosphorylation-of",
    "controls-state-change-of", "controls-transport-of", "in-complex-with",
    "interacts-with"),
  ...
)
```

Arguments

`interaction_types`	Character vector with PathwayCommons interaction types. Please refer to the default value and the PathwayCommons webpage.
`...`	Ignored.

Value

A network data frame (tibble) with signaling interactions suitable for use with NicheNet.

Examples

```
# use only the "controls-transport-of" interactions:
pc_signaling_network <- nichenet_signaling_network_pathwaycommons(
    interaction_types = 'controls-transport-of'
)
```

`nichenet_signaling_network_vinayagam`

NicheNet signaling network from Vinayagam

Description

Builds signaling network prior knowledge for NicheNet using Vinayagam 2011 Supplementary Table S6. Find out more at https://doi.org/10.1126/scisignal.2001699.

Usage

```
nichenet_signaling_network_vinayagam(...)
```

Arguments

`...`	Ignored.

Value

A network data frame (tibble) with signaling interactions suitable for use with NicheNet.

Examples

```
vi_signaling_network <- nichenet_signaling_network_vinayagam()
```

`nichenet_test` *Run the NicheNet pipeline with a little dummy network*

Description

Loads a tiny network and runs the NicheNet pipeline with low number of iterations in the optimization process. This way the pipeline runs in a reasonable time in order to test the code. Due to the random subsampling disconnected networks might be produced sometimes. If you see an error like "Error in if (sd(prediction_vector) == 0) ... missing value where TRUE/FALSE needed", the random subsampled input is not appropriate. In this case just interrupt and call again. This test ensures the computational integrity of the pipeline. If it fails during the optimization process, try to start it over several times, even restarting R. The unpredictability is related to `mlrMBO` and `nichenetr` not being prepared to handle certain conditions, and it's also difficult to find out which conditions lead to which errors. At least 3 different errors appear time to time, depending on the input. It also seems like restarting R sometimes helps, suggesting that the entire system might be somehow stateful. You can ignore the `Parallelization was not stopped` warnings on repeated runs.

Usage

```
nichenet_test(...)
```

Arguments

`...`	Passed to `nichenet_main`.

Value

A named list with the intermediate and final outputs of the pipeline: 'networks', 'expression', 'optimized_parameters', 'weighted_networks' and 'ligand_target_matrix'.

Examples

```
## Not run:
nnt <- nichenet_test()

## End(Not run)
```

`nichenet_workarounds` *Workarounds using NicheNet without attaching the package*

Description

NicheNet requires the availability of some lazy loaded external data which are not available if the package is not loaded and attached. Also, the `BBmisc::convertToShortString` used for error reporting in `mlrMBO::evalTargetFun.OptState` is patched here to print longer error messages. Maybe it's a better solution to attach `nichenetr` before running the NicheNet pipeline. Alternatively you can try to call this function in the beginning. Why we don't call this automatically is just because we don't want to load datasets from another package without the user knowing about it.

Usage

```
nichenet_workarounds()
```

Value

Returns `NULL`.

Examples

```
## Not run:
nichenet_workarounds()

## End(Not run)
```

`obo_parser` *Generic OBO parser*

Description

Reads the contents of an OBO file and processes it into data frames or a list based data structure.

Usage

```
obo_parser(
  path,
 relations = c("is_a", "part_of", "occurs_in", "regulates", "positively_regulates",
    "negatively_regulates"),
  shorten_namespace = TRUE,
  tables = TRUE
)
```

Arguments

`path`	Path to the OBO file.
`relations`	Character vector: process only these relations.
`shorten_namespace`	Logical: shorten the namespace to a single letter code (as usual for Gene Ontology, e.g. cellular_component = "C").
`tables`	Logical: return data frames (tibbles) instead of nested lists.

Value

A list with the following elements: 1) "names" a list with terms as names and names as values; 2) "namespaces" a list with terms as names and namespaces as values; 3) "relations" a list with relations between terms: terms are keys, values are lists with relations as names and character vectors of related terms as values; 4) "subsets" a list with terms as keys and character vectors of subset names as values (or `NULL` if the term does not belong to any subset); 5) "obsolete" character vector with all the terms labeled as obsolete. If the `tables` parameter is `TRUE`, "names", "namespaces", "relations" and "subsets" will be data frames (tibbles).

See Also

- relations_list_to_table
- relations_table_to_list
- swap_relations

Examples

```
goslim_url <-
    "http://current.geneontology.org/ontology/subsets/goslim_generic.obo"
path <- tempfile()
httr::GET(goslim_url, httr::write_disk(path, overwrite = TRUE))
obo <- obo_parser(path, tables = FALSE)
unlink(path)
names(obo)
# [1] "names"      "namespaces" "relations"  "subsets"    "obsolete"
head(obo$relations, n = 2)
# $`GO:0000001`
# $`GO:0000001`$is_a
# [1] "GO:0048308" "GO:0048311"
#
# $`GO:0000002`
# $`GO:0000002`$is_a
# [1] "GO:0007005"
```

`oma_code` *Orthologous Matrix (OMA) codes of organisms*

Description

Note: OMA species codes are whenever possible identical to UniProt codes.

Usage

```
oma_code(name)
```

Arguments

`name`	Vector with any kind of organism name or identifier, can be also mixed type.

Value

A character vector with the Orthologous Matrix (OMA) codes of the organisms.

See Also

- ncbi_taxid
- latin_name
- ensembl_name
- common_name

Examples

```
oma_code(c(10090, "cjacchus", "Vicugna pacos"))
# [1] "MOUSE" "CALJA" "VICPA"
```

`oma_organisms` *Organism identifiers from the Orthologous Matrix*

Description

Organism identifiers from the Orthologous Matrix

Usage

```
oma_organisms()
```

Value

A data frame with organism identifiers.

See Also

`ensembl_organisms`

Examples

```
oma_organisms()
```

`oma_pairwise` *Orthologous gene pairs between two organisms*

Description

From the web API of Orthologous Matrix (OMA). Items which could not be translated to ‘id_type‘ (but present in the data with their internal OMA IDs) are removed.

Usage

```
oma_pairwise(
  organism_a = "human",
  organism_b = "mouse",
  id_type = "uniprot",
  mappings = c("1:1", "1:m", "n:1", "n:m"),
  only_ids = TRUE
)
```

Arguments

`organism_a`	Name or identifier of an organism.
`organism_b`	Name or identifier of another organism.
`id_type`	The gene or protein identifier to use in the table. For a list of supported ID types see ‘omnipathr.envid_typesoma‘. In addition, "genesymbol" is supported, in this case `oma_pairwise_genesymbols` will be called automatically.
`mappings`	Character vector: control ambiguous mappings: • 1:1 - unambiguous • 1:m - one-to-many • n:1 - many-to-one • n:m - many-to-many
`only_ids`	Logical: include only the two identifier columns, not the mapping type and the orthology group columns.

Value

A data frame with orthologous gene pairs.

Examples

```
oma_pairwise("human", "mouse", "uniprot")
# # A tibble: 21,753 × 4
#    id_organism_a id_organism_b mapping oma_group
#    <chr>         <chr>         <chr>       <dbl>
#  1 Q15326        Q8R5C8        1:1       1129380
#  2 Q9Y2E4        B2RQ71        1:1        681224
#  3 Q92615        Q6A0A2        1:1       1135087
#  4 Q9BZE4        Q99ME9        1:1       1176239
#  5 Q9BXS1        Q8BFZ6        1:m            NA
# # . . .  with 21,743 more rows
```

`oma_pairwise_genesymbols`

Orthologous pairs of gene symbols between two organisms

Description

The Orthologous Matrix (OMA), a resource of orthologous relationships between genes, doesn’t provide gene symbols, the identifier preferred in many bioinformatics pipelines. Hence this function wraps `oma_pairwise` by translating the identifiers used in OMA to gene symbols. Items that can not be translated to ‘id_type‘ (but present in the data with their internal OMA IDs) will be removed. Then, in this function we translate the identifiers to gene symbols.

Usage

```
oma_pairwise_genesymbols(
  organism_a = "human",
  organism_b = "mouse",
  oma_id_type = "uniprot_entry",
  mappings = c("1:1", "1:m", "n:1", "n:m"),
  only_ids = TRUE
)
```

Arguments

`organism_a`	Name or identifier of an organism.
`organism_b`	Name or identifier of another organism.
`oma_id_type`	Character: the gene or protein identifier to be queried from OMA. These IDs will be translated to ‘id_type‘.
`mappings`	Character vector: control ambiguous mappings: • 1:1 - unambiguous • 1:m - one-to-many • n:1 - many-to-one • n:m - many-to-many
`only_ids`	Logical: include only the two identifier columns, not the mapping type and the orthology group columns.

Value

A data frame with orthologous gene pairs.

Examples

```
oma_pairwise_genesymbols("human", "mouse")
```

`oma_pairwise_translated` *Orthologous pairs between two organisms for ID types not supported by OMA*

Description

The Orthologous Matrix (OMA), a resource of orthologous relationships between genes, doesn’t provide gene symbols, the identifier preferred in many bioinformatics pipelines. Hence this function wraps `oma_pairwise` by translating the identifiers used in OMA to gene symbols. Items that can not be translated to ‘id_type‘ (but present in the data with their internal OMA IDs) will be removed. Then, in this function we translate the identifiers to the desired ID type.

Usage

```
oma_pairwise_translated(
  organism_a = "human",
  organism_b = "mouse",
  id_type = "uniprot",
  oma_id_type = "uniprot_entry",
  mappings = c("1:1", "1:m", "n:1", "n:m"),
  only_ids = TRUE
)
```

Arguments

`organism_a`	Name or identifier of an organism.
`organism_b`	Name or identifier of another organism.
`id_type`	The gene or protein identifier to use in the table. For a list of supported ID types see ‘omnipathr.envid_typesoma‘. These are the identifiers that will be translated to gene symbols.
`oma_id_type`	Character: the gene or protein identifier to be queried from OMA. These IDs will be translated to ‘id_type‘.
`mappings`	Character vector: control ambiguous mappings: • 1:1 - unambiguous • 1:m - one-to-many • n:1 - many-to-one • n:m - many-to-many
`only_ids`	Logical: include only the two identifier columns, not the mapping type and the orthology group columns.

Value

A data frame with orthologous gene pairs.

Examples

```
oma_pairwise_translated("human", "mouse")
```

`omnipath-interactions` *Molecular interactions from OmniPath*

Description

The functions listed here all download pairwise, causal molecular interactions from the `https://omnipathdb.org/interactions` endpoint of the OmniPath web service. They are different only in the type of interactions and the kind of resources and data they have been compiled from. A complete list of these functions is available below, these cover the interaction datasets and types currently available in OmniPath:

Interactions from the `https://omnipathdb.org/interactions` endpoint of the OmniPath web service. By default, it downloads only the "omnipath" dataset, which corresponds to the curated causal interactions described in Turei et al. 2016.

Imports interactions from the ‘omnipath‘ dataset of OmniPath, a dataset that inherits most of its design and contents from the original OmniPath core from the 2016 publication. This dataset consists of about 40k interactions.

Imports the dataset from: `https://omnipathdb.org/interactions?datasets=pathwayextra`, which contains activity flow interactions without literature reference. The activity flow interactions supported by literature references are part of the ‘omnipath‘ dataset.

Imports the dataset from: `https://omnipathdb.org/interactions?datasets=kinaseextra`, which contains enzyme-substrate interactions without literature reference. The enzyme-substrate interactions supported by literature references are part of the ‘omnipath‘ dataset.

Imports the dataset from: `https://omnipathdb.org/interactions?datasets=ligrecextra`, which contains ligand-receptor interactions without literature reference. The ligand-receptor interactions supported by literature references are part of the ‘omnipath‘ dataset.

Imports interactions from all post-translational datasets of OmniPath. The datasets are "omnipath", "kinaseextra", "pathwayextra" and "ligrecextra".

Imports the dataset from: `https://omnipathdb.org/interactions?datasets=dorothea` which contains transcription factor (TF)-target interactions from DoRothEA `https://github.com/saezlab/DoRothEA` DoRothEA is a comprehensive resource of transcriptional regulation, consisting of 16 original resources, in silico TFBS prediction, gene expression signatures and ChIP-Seq binding site analysis.

Imports the dataset from: `https://omnipathdb.org/interactions?datasets=tf_target`, which contains transcription factor-target protein coding gene interactions. Note: this is not the only TF-target dataset in OmniPath, ‘dorothea‘ is the other one and the ‘tf_mirna‘ dataset provides TF-miRNA gene interactions.

Imports the dataset from: `https://omnipathdb.org/interactions?datasets=tf_target,dorothea`, which contains transcription factor-target protein coding gene interactions.

CollecTRI is a comprehensive resource of transcriptional regulation, published in 2023, consisting of 14 resources and original literature curation.

Imports the dataset from: `https://omnipathdb.org/interactions?datasets=mirnatarget`, which contains miRNA-mRNA interactions.

Imports the dataset from: `https://omnipathdb.org/interactions?datasets=tf_mirna`, which contains transcription factor-miRNA gene interactions

Imports the dataset from: `https://omnipathdb.org/interactions?datasets=lncrna_mrna`, which contains lncRNA-mRNA interactions

Imports the dataset from: `https://omnipathdb.org/interactions?datasets=small_molecule`, which contains small molecule-protein interactions. Small molecules can be metabolites, intrinsic ligands or drug compounds.

Usage

```
omnipath_interactions(...)

omnipath(...)

pathwayextra(...)

kinaseextra(...)

ligrecextra(...)
```

```
post_translational(...)

dorothea(dorothea_levels = c("A", "B"), ...)

tf_target(...)

transcriptional(dorothea_levels = c("A", "B"), ...)

collectri(...)

mirna_target(...)

tf_mirna(...)

lncrna_mrna(...)

small_molecule(...)

all_interactions(
  dorothea_levels = c("A", "B"),
  types = NULL,
  fields = NULL,
  exclude = NULL,
  ...
)
```

Arguments

`...`	Arguments passed on to `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query`, `omnipath_query` `organism` Character or integer: name or NCBI Taxonomy ID of the organism. OmniPath is built of human data, and the web service provides orthology translated interactions and enzyme-substrate relationships for mouse and rat. For other organisms and query types, orthology translation will be called automatically on the downloaded human data before returning the result. `resources` Character vector: name of one or more resources. Restrict the data to these resources. For a complete list of available resources, call the '<query_type>_resources' functions for the query type of interst. `datasets` Character vector: name of one or more datasets. In the interactions query type a number of datasets are available. The default is caled "omnipath", and corresponds to the curated causal signaling network published in the 2016 OmniPath paper. `genesymbols` Character or logical: TRUE or FALS or "yes" or "no". Include the 'genesymbols' column in the results. OmniPath uses UniProt IDs as the primary identifiers, gene symbols are optional. `default_fields` Logical: if TRUE, the default fields will be included. `silent` Logical: if TRUE, no messages will be printed. By default a summary message is printed upon successful download. `logicals` Character vector: fields to be cast to logical.

- `format` Character: if "json", JSON will be retrieved and processed into a nested list; any other value will return data frame.
- `download_args` List: parameters to pass to the download function, which is ‘readr::read_tsv‘ by default, and ‘jsonlite::safe_load‘.
- `references_by_resource` Logical: if TRUE,, in the ‘references‘ column the PubMed IDs will be prefixed with the names of the resources they are coming from. If FALSE, the ‘references‘ column will be a list of unique PubMed IDs.
- `add_counts` Logical: if TRUE, the number of references and number of resources for each record will be added to the result.
- `license` Character: license restrictions. By default, data from resources allowing "academic" use is returned by OmniPath. If you use the data for work in a company, you can provide "commercial" or "for-profit", which will restrict the data to those records which are supported by resources that allow for-profit use.
- `password` Character: password for the OmniPath web service. You can provide a special password here which enables the use of ‘license = "ignore"‘ option, completely bypassing the license filter.
- `json_param` List: parameters to pass to the ‘jsonlite::fromJSON‘ when processing JSON columns embedded in the downloaded data. Such columns are "extra_attrs" and "evidences". These are optional columns which provide a lot of extra details about interactions.
- `strict_evidences` Logical: reconstruct the "sources" and "references" columns of interaction data frames based on the "evidences" column, strictly filtering them to the queried datasets and resources. Without this, the "sources" and "references" fields for each record might contain information for datasets and resources other than the queried ones, because the downloaded records are a result of a simple filtering of an already integrated data frame.
- `genesymbol_resource` Character: "uniprot" (default) or "ensembl". The OmniPath web service uses the primary gene symbols as provided by UniProt. By passing "ensembl" here, the UniProt gene symbols will be replaced by the ones used in Ensembl. This translation results in a loss of a few records, and multiplication of another few records due to ambiguous translation.
- `cache` Logical: use caching, load data from and save to the. The cache directory by default belongs to the user, located in the user’s default cache directory, and named "OmnipathR". Find out about it by `getOption("omnipathr.cachedir")`. Can be changed by `omnipath_set_cachedir`.

`dorothea_levels`	The confidence levels of the dorothea interactions (TF-target) which range from A to D. Set to A and B by default.
`types`	Character: interaction types, such as "transcriptional", "post_transcriptional", "post_translational", etc.
`fields`	Character: additional fields (columns) to be included in the result. For a list of available fields, see `query_info`.
`exclude`	Character: names of datasets or resource to be excluded from the result. By deafult, the records supported by only these resources or datasets will be removed from the output. If `strict_evidences = TRUE`, the resource, reference and causality information in the data frame will be reconstructed to remove all information coming from the excluded resources.

Details

Post-translational (protein-protein, PPI) interactions

- `omnipath`: the OmniPath data as defined in the 2016 paper, an arbitrary optimum between coverage and quality. This dataset contains almost entirely causal (stimulatory or inhibitory; i.e. activity flow , according to the SBGN standard), physical interactions between pairs of proteins, curated by experts from the literature.
- `pathwayextra`: activity flow interactions without literature references.
- `kinaseextra`: enzyme-substrate interactions without literature references.
- `ligrecextra`: ligand-receptor interactions without literature references.
- `post_translational`: all post-translational (protein-protein, PPI) interactions; this is the combination of the *omnipath*, *pathwayextra*, *kinaseextra* and *ligrecextra* datasets.

TF-target (gene regulatory, GRN) interactions

- `collectri`: transcription factor (TF)-target interactions from CollecTRI.
- `dorothea`: transcription factor (TF)-target interactions from DoRothEA
- `tf_target`: transcription factor (TF)-target interactions from other resources
- `transcriptional`: all transcription factor (TF)-target interactions; this is the combination of the *collectri*, *dorothea* and *tf_target* datasets.

Post-transcriptional (miRNA-target) and other RNA related interactions

In these datasets we intend to collect the literature curated resources, hence we don't include some of the most well known large databases if those are based on predictions or high-throughput assays.

- `mirna_target`: miRNA-mRNA interactions
- `tf_mirna`: TF-miRNA interactions
- `lncrna_mrna`: lncRNA-mRNA interactions

Other interaction access functions

- `small_molecule`: interactions between small molecules and proteins. Currently this is a small, experimental dataset that includes drug-target, ligand-receptor, enzyme-metabolite and other interactions. In the future this will be largely expanded and divided into multiple datasets.
- `all_interactions`: all the interaction datasets combined.

Value

A dataframe of molecular interactions.

A dataframe of literature curated, post-translational signaling interactions.

A dataframe containing activity flow interactions between proteins without literature reference

A dataframe containing enzyme-substrate interactions without literature reference

A dataframe containing ligand-receptor interactions including the ones without literature references

A dataframe containing post-translational interactions

A data frame of TF-target interactions from DoRothEA.

A dataframe containing TF-target interactions

A dataframe containing TF-target interactions.

A dataframe of TF-target interactions.

A dataframe containing miRNA-mRNA interactions

A dataframe containing TF-miRNA interactions

A dataframe containing lncRNA-mRNA interactions

A dataframe of small molecule-protein interactions

A dataframe containing all the datasets in the interactions query

See Also

- `interaction_resources`
- `interaction_graph`
- `print_interactions`
- `annotated_network`

- `omnipath_interactions`
- `post_translational`
- `interaction_resources`
- `all_interactions`
- `interaction_graph`
- `print_interactions`

Examples

```
op <- omnipath(resources = c("CA1", "SIGNOR", "SignaLink3"))
op

interactions = omnipath_interactions(
    resources = "SignaLink3",
    organism = 9606
)

pathways <- omnipath()
pathways

interactions <-
    pathwayextra(
        resources = c("BioGRID", "IntAct"),
        organism = 9606
    )

kinase_substrate <-
   kinaseextra(
       resources = c('PhosphoPoint', 'PhosphoSite'),
       organism = 9606
   )

ligand_receptor <- ligrecextra(
    resources = c('HPRD', 'Guide2Pharma'),
    organism = 9606
)
```

```
interactions <- post_translational(resources = "BioGRID")

dorothea_grn <- dorothea(
    resources = c('DoRothEA', 'ARACNe-GTEx_DoRothEA'),
    organism = 9606,
    dorothea_levels = c('A', 'B', 'C')
)
dorothea_grn

interactions <- tf_target(resources = c("DoRothEA", "SIGNOR"))

grn <- transcriptional(resources = c("PAZAR", "ORegAnno", "DoRothEA"))
grn

collectri_grn <- collectri()
collectri_grn

interactions <- mirna_target( resources = c("miRTarBase", "miRecords"))

interactions <- tf_mirna(resources = "TransmiR")

interactions <- lncrna_mrna(resources = c("ncRDeathDB"))

# What are the targets of aspirin?
interactions <- small_molecule(sources = "ASPIRIN")
# The prostaglandin synthases:
interactions

interactions <- all_interactions(
    resources = c("HPRD", "BioGRID"),
    organism = 9606
)
```

`OmnipathR` *The OmnipathR package*

Description

OmnipathR is an R package built to provide easy access to the data stored in the OmniPath web service:

https://omnipathdb.org/

And a number of other resources, such as BioPlex, ConsensusPathDB, EVEX, Guide to Pharmacology (IUPHAR/BPS), Harmonizome, HTRIdb, InWeb InBioMap, KEGG Pathway, Pathway Commons, Ramilowski et al. 2015, RegNetwork, ReMap, TF census, TRRUST and Vinayagam et al. 2011.

The OmniPath web service implements a very simple REST style API. This package make requests by the HTTP protocol to retreive the data. Hence, fast Internet access is required for a propser use of OmnipathR.

The package also provides some utility functions to filter, analyse and visualize the data. Furthermore, OmnipathR features a close integration with the NicheNet method for ligand activity prediction from transcriptomics data, and its R implementation nichenetr (available in CRAN).

Author(s)

Alberto Valdeolivas <<`alvaldeolivas@gmail`>> and Denes Turei <<`turei.denes@gmail.com`>> and Attila Gabor <<`gaborattila87@gmail.com`>>

See Also

Useful links:

- `https://r.omnipathdb.org/`
- Report bugs at `https://github.com/saezlab/OmnipathR/issues`

Examples

```
## Not run:
# Download post-translational modifications:
enzsub <- enzyme_substrate(resources = c("PhosphoSite", "SIGNOR"))

# Download protein-protein interactions
interactions <- omnipath(resources = "SignaLink3")

# Convert to igraph objects:
enzsub_g <- enzsub_graph(enzsub = enzsub)
OPI_g <- interaction_graph(interactions = interactions)

# Print some interactions:
print_interactions(head(enzsub))

# interactions with references:
print_interactions(tail(enzsub), writeRefs = TRUE)

# find interactions between kinase and substrate:
print_interactions(dplyr::filter(ptms,enzyme_genesymbol=="MAP2K1",
   substrate_genesymbol=="MAPK3"))

# find shortest paths on the directed network between proteins
print_path_es(shortest_paths(OPI_g, from = "TYRO3", to = "STAT3",
   output = 'epath')$epath[[1]], OPI_g)

# find all shortest paths between proteins
print_path_vs(
    all_shortest_paths(
        enzsub_g,
        from = "SRC",
        to = "STAT1"
    )$res,
    enzsub_g
)

## End(Not run)
```

`omnipath_cache_autoclean` *Keeps only the latest versions of complete downloads*

Description

Removes the old versions, the failed downloads and the files in the cache directory which are missing from the database. For more flexible operations use `omnipath_cache_remove` and `omnipath_cache_clean`.

Usage

```
omnipath_cache_autoclean()
```

Value

Invisibl returns the cache database (list of cache records).

Examples

```
## Not run:
omnipath_cache_autoclean()

## End(Not run)
```

`omnipath_cache_clean` *Removes the items from the cache directory which are unknown by the cache database*

Description

Removes the items from the cache directory which are unknown by the cache database

Usage

```
omnipath_cache_clean()
```

Value

Returns ‘NULL‘.

Examples

```
omnipath_cache_clean()
```

`omnipath_cache_clean_db` *Removes the cache database entries without existing files*

Description

Removes the cache database entries without existing files

Usage

```
omnipath_cache_clean_db(...)
```

Arguments

`...`	Ignored.

Value

Returns ‘NULL‘.

Examples

```
omnipath_cache_clean_db()
```

`omnipath_cache_download_ready` *Sets the download status to ready for a cache item*

Description

Sets the download status to ready for a cache item

Usage

```
omnipath_cache_download_ready(version, key = NULL)
```

Arguments

`version`	Version of the cache item. If does not exist a new version item will be created
`key`	Key of the cache item

Value

Character: invisibly returns the version number of the cache version item.

Examples

```
bioc_url <- 'https://bioconductor.org/'
# request a new version item (or retrieve the latest)
new_version <- omnipath_cache_latest_or_new(url = bioc_url)
# check if the version item is not a finished download
new_version$status
# [1] "unknown"
# download the file
httr::GET(bioc_url, httr::write_disk(new_version$path, overwrite = TRUE))
# report to the cache database that the download is ready
omnipath_cache_download_ready(new_version)
# now the status is ready:
version <- omnipath_cache_latest_or_new(url = bioc_url)
version$status
# "ready"
version$dl_finished
# [1] "2021-03-09 16:48:38 CET"
omnipath_cache_remove(url = bioc_url) # cleaning up
```

`omnipath_cache_filter_versions`

Filters the versions from one cache record

Description

Filters the versions based on multiple conditions: their age and status

Usage

```
omnipath_cache_filter_versions(
  record,
  latest = FALSE,
  max_age = NULL,
  min_age = NULL,
  status = CACHE_STATUS$READY
)
```

Arguments

`record`	A cache record
`latest`	Return the most recent version
`max_age`	The maximum age in days (e.g. 5: 5 days old or more recent)
`min_age`	The minimum age in days (e.g. 5: 5 days old or older)
`status`	Character vector with status codes. By default only the versions with 'ready' (completed download) status are selected

Value

Character vector with version IDs, NA if no version satisfies the conditions.

Examples

```
# creating an example cache record
bioc_url <- 'https://bioconductor.org/'
version <- omnipath_cache_latest_or_new(url = bioc_url)
httr::GET(bioc_url, httr::write_disk(version$path, overwrite = TRUE))
omnipath_cache_download_ready(version)
record <- dplyr::first(omnipath_cache_search('biocond'))

# only the versions with status "ready"
version_numbers <- omnipath_cache_filter_versions(record, status = 'ready')
omnipath_cache_remove(url = bioc_url) # cleaning up
```

`omnipath_cache_get` *Retrieves one item from the cache directory*

Description

Retrieves one item from the cache directory

Usage

```
omnipath_cache_get(
  key = NULL,
  url = NULL,
  post = NULL,
  payload = NULL,
  create = TRUE,
  ...
)
```

Arguments

`key`	The key of the cache record
`url`	URL pointing to the resource
`post`	HTTP POST parameters as a list
`payload`	HTTP data payload
`create`	Create a new entry if doesn't exist yet
`...`	Passed to `omnipath_cache_record` (internal function)

Value

Cache record: an existing record if the entry already exists, otherwise a newly created and inserted record

Examples

```
# create an example cache record
bioc_url <- 'https://bioconductor.org/'
version <- omnipath_cache_latest_or_new(url = bioc_url)
omnipath_cache_remove(url = bioc_url) # cleaning up

# retrieve the cache record
record <- omnipath_cache_get(url = bioc_url)
record$key
# [1] "41346a00fb20d2a9df03aa70cf4d50bf88ab154a"
record$url
# [1] "https://bioconductor.org/"
```

`omnipath_cache_key` *Generates a hash which identifies an element in the cache database*

Description

Generates a hash which identifies an element in the cache database

Usage

```
omnipath_cache_key(url, post = NULL, payload = NULL)
```

Arguments

`url`	Character vector with URLs
`post`	List with the HTTP POST parameters or a list of lists if the url vector is longer than 1. NULL for queries without POST parameters.
`payload`	HTTP data payload. List with multiple items if the url vector is longer than 1. NULL for queries without data.

Value

Character vector of cache record keys.

Examples

```
bioc_url <- 'https://bioconductor.org/'
omnipath_cache_key(bioc_url)
# [1] "41346a00fb20d2a9df03aa70cf4d50bf88ab154a"
```

`omnipath_cache_latest_or_new` *The latest or a new version of a cache record*

Description

Looks up a record in the cache and returns its latest valid version. If the record doesn't exist or no valid version available, creates a new one.

Usage

```
omnipath_cache_latest_or_new(
  key = NULL,
  url = NULL,
  post = NULL,
  payload = NULL,
  create = TRUE,
  ...
)
```

Arguments

`key`	The key of the cache record
`url`	URL pointing to the resource
`post`	HTTP POST parameters as a list
`payload`	HTTP data payload
`create`	Logical: whether to create and return a new version. If FALSE only the latest existing valid version is returned, if available.
`...`	Passed to `omnipath_cache_get`

Value

A cache version item.

Examples

```
## Not run:
# retrieve the latest version of the first cache record
# found by the search keyword "bioplex"
latest_bioplex <-
    omnipath_cache_latest_or_new(
        names(omnipath_cache_search('bioplex'))[1]
    )

latest_bioplex$dl_finished
# [1] "2021-03-09 14:28:50 CET"
latest_bioplex$path
# [1] "/home/denes/.cache/OmnipathR/378e0def2ac97985f629-1.rds"

## End(Not run)
```

```
# create an example cache record
bioc_url <- 'https://bioconductor.org/'
version <- omnipath_cache_latest_or_new(url = bioc_url)
omnipath_cache_remove(url = bioc_url) # cleaning up
```

`omnipath_cache_latest_version` *Finds the most recent version in a cache record*

Description

Finds the most recent version in a cache record

Usage

```
omnipath_cache_latest_version(record)
```

Arguments

`record`	A cache record

Value

Character: the version ID with the most recent download finished time

---

`omnipath_cache_load` *Loads an R object from the cache*

Description

Loads the object from RDS format.

Usage

```
omnipath_cache_load(
  key = NULL,
  version = NULL,
  url = NULL,
  post = NULL,
  payload = NULL
)
```

Arguments

`key`	Key of the cache item
`version`	Version of the cache item. If does not exist or NULL, the latest version will be retrieved
`url`	URL of the downloaded resource
`post`	HTTP POST parameters as a list
`payload`	HTTP data payload

Value

Object loaded from the cache RDS file.

See Also

omnipath_cache_save

Examples

```
url <- paste0(
    'https://omnipathdb.org/intercell?resources=Adhesome,Almen2009,',
    'Baccin2019,CSPA,CellChatDB&license=academic'
)
result <- read.delim(url, sep = '\t')
omnipath_cache_save(result, url = url)
# works only if you have already this item in the cache
intercell_data <- omnipath_cache_load(url = url)
class(intercell_data)
# [1] "data.frame"
nrow(intercell_data)
# [1] 16622
attr(intercell_data, 'origin')
# [1] "cache"

# basic example of saving and loading to and from the cache:
bioc_url <- 'https://bioconductor.org/'
bioc_html <- readChar(url(bioc_url), nchars = 99999)
omnipath_cache_save(bioc_html, url = bioc_url)
bioc_html <- omnipath_cache_load(url = bioc_url)
```

`omnipath_cache_move_in`

Moves an existing file into the cache

Description

Either the key or the URL (with POST and payload) must be provided.

Usage

```
omnipath_cache_move_in(
  path,
  key = NULL,
  version = NULL,
  url = NULL,
  post = NULL,
  payload = NULL,
  keep_original = FALSE
)
```

Arguments

`path`	Path to the source file
`key`	Key of the cache item
`version`	Version of the cache item. If does not exist a new version item will be created
`url`	URL of the downloaded resource
`post`	HTTP POST parameters as a list
`payload`	HTTP data payload
`keep_original`	Whether to keep or remove the original file

Value

Character: invisibly returns the version number of the cache version item.

See Also

`omnipath_cache_save`

Examples

```
path <- tempfile()
saveRDS(rnorm(100), file = path)
omnipath_cache_move_in(path, url = 'the_download_address')

# basic example of moving a file to the cache:

bioc_url <- 'https://bioconductor.org/'
html_file <- tempfile(fileext = '.html')
httr::GET(bioc_url, httr::write_disk(html_file, overwrite = TRUE))
omnipath_cache_move_in(path = html_file, url = bioc_url)
omnipath_cache_remove(url = bioc_url) # cleaning up
```

`omnipath_cache_remove` *Removes contents from the cache directory*

Description

According to the parameters, it can remove contents older than a certain age, or contents having a more recent version, one specific item, or wipe the entire cache.

Usage

```
omnipath_cache_remove(key = NULL, url = NULL, post = NULL,
    payload = NULL, max_age = NULL, min_age = NULL, status = NULL,
    only_latest = FALSE, wipe = FALSE, autoclean = TRUE)
```

Arguments

`key`	The key of the cache record
`url`	URL pointing to the resource
`post`	HTTP POST parameters as a list
`payload`	HTTP data payload
`max_age`	Age of cache items in days. Remove everything that is older than this age
`min_age`	Age of cache items in days. Remove everything more recent than this age
`status`	Remove items having any of the states listed here
`only_latest`	Keep only the latest version
`wipe`	Logical: if TRUE, removes all files from the cache and the cache database. Same as calling `omnipath_cache_wipe`.
`autoclean`	Remove the entries about failed downloads, the files in the cache directory which are missing from the cache database, and the entries without existing files in the cache directory

Value

Invisibly returns the cache database (list of cache records).

See Also

- `omnipath_cache_wipe`
- `omnipath_cache_clean`
- `omnipath_cache_autoclean`

Examples

```
## Not run:
# remove all cache data from the BioPlex database
cache_records <- omnipath_cache_search(
    'bioplex',
    ignore.case = TRUE
)
omnipath_cache_remove(names(cache_records))

# remove a record by its URL
regnetwork_url <- 'http://www.regnetworkweb.org/download/human.zip'
omnipath_cache_remove(url = regnetwork_url)

# remove all records older than 30 days
omnipath_cache_remove(max_age = 30)

# for each record, remove all versions except the latest
omnipath_cache_remove(only_latest = TRUE)

## End(Not run)

bioc_url <- 'https://bioconductor.org/'
version <- omnipath_cache_latest_or_new(url = bioc_url)
httr::GET(bioc_url, httr::write_disk(version$path, overwrite = TRUE))
omnipath_cache_download_ready(version)
```

```
key <- omnipath_cache_key(bioc_url)
omnipath_cache_remove(key = key)
```

`omnipath_cache_save` *Saves an R object to the cache*

Description

Exports the object in RDS format, creates new cache record if necessary.

Usage

```
omnipath_cache_save(
  data,
  key = NULL,
  version = NULL,
  url = NULL,
  post = NULL,
  payload = NULL
)
```

Arguments

`data`	An object
`key`	Key of the cache item
`version`	Version of the cache item. If does not exist a new version item will be created
`url`	URL of the downloaded resource
`post`	HTTP POST parameters as a list
`payload`	HTTP data payload

Value

Returns invisibly the data itself.

Invisibly returns the ‘data‘.

See Also

`omnipath_cache_move_in`

Examples

```
mydata <- data.frame(a = c(1, 2, 3), b = c('a', 'b', 'c'))
omnipath_cache_save(mydata, url = 'some_dummy_address')
from_cache <- omnipath_cache_load(url = 'some_dummy_address')
from_cache
#   a b
# 1 1 a
# 2 2 b
# 3 3 c
```

```
attr(from_cache, 'origin')
# [1] "cache"

# basic example of saving and loading to and from the cache:
bioc_url <- 'https://bioconductor.org/'
bioc_html <- readChar(url(bioc_url), nchars = 99999)
omnipath_cache_save(bioc_html, url = bioc_url)
bioc_html <- omnipath_cache_load(url = bioc_url)
```

`omnipath_cache_search` *Searches for cache items*

Description

Searches the cache records by matching the URL against a string or regexp.

Usage

```
omnipath_cache_search(pattern, ...)
```

Arguments

`pattern`	String or regular expression.
`...`	Passed to `grep`

Value

List of cache records matching the pattern.

Examples

```
# find all cache records from the BioPlex database
bioplex_cache_records <- omnipath_cache_search(
    'bioplex',
    ignore.case = TRUE
)
```

`omnipath_cache_set_ext` *Sets the file extension for a cache record*

Description

Sets the file extension for a cache record

Usage

```
omnipath_cache_set_ext(key, ext)
```

Arguments

`key`	Character: key for a cache item, alternatively a version entry.
`ext`	Character: the file extension, e.g. "zip".

Value

Returns ‘NULL‘.

Examples

```
bioc_url <- 'https://bioconductor.org/'
version <- omnipath_cache_latest_or_new(url = bioc_url)
version$path
# [1] "/home/denes/.cache/OmnipathR/41346a00fb20d2a9df03-1"
httr::GET(bioc_url, httr::write_disk(version$path, overwrite = TRUE))
key <- omnipath_cache_key(url = bioc_url)
omnipath_cache_set_ext(key = key, ext = 'html')
version <- omnipath_cache_latest_or_new(url = bioc_url)
version$path
# [1] "/home/denes/.cache/OmnipathR/41346a00fb20d2a9df03-1.html"
record <- omnipath_cache_get(url = bioc_url)
record$ext
# [1] "html"
omnipath_cache_remove(url = bioc_url) # cleaning up
```

`omnipath_cache_update_status`

Updates the status of an existing cache record

Description

Updates the status of an existing cache record

Usage

```
omnipath_cache_update_status(key, version, status,
    dl_finished = NULL)
```

Arguments

`key`	Key of the cache item
`version`	Version of the cache item. If does not exist a new version item will be created
`status`	The updated status value
`dl_finished`	Timestamp for the time when download was finished, if ‘NULL‘ the value remains unchanged

Value

Character: invisibly returns the version number of the cache version item.

Examples

```
bioc_url <- 'https://bioconductor.org/'
latest_version <- omnipath_cache_latest_or_new(url = bioc_url)
key <- omnipath_cache_key(bioc_url)
omnipath_cache_update_status(
    key = key,
    version = latest_version$number,
    status = 'ready',
    dl_finished = Sys.time()
)
omnipath_cache_remove(url = bioc_url) # cleaning up
```

`omnipath_cache_wipe` *Permanently removes all the cache contents*

Description

After this operation the cache directory will be completely empty, except an empty cache database file.

Usage

```
omnipath_cache_wipe(...)
```

Arguments

`...`	Ignored.

Value

Returns ‘NULL‘.

See Also

`omnipath_cache_remove`

Examples

```
## Not run:
omnipath_cache_wipe()
# the cache is completely empty:
print(omnipathr.env$cache)
# list()
list.files(omnipath_get_cachedir())
# [1] "cache.json"

## End(Not run)
```

`omnipath_config_path` *Current config file path of OmnipathR*

Description

Current config file path of OmnipathR

Current config file path for a certain package

Usage

```
omnipath_config_path(user = FALSE)

config_path(user = FALSE, pkg = "OmnipathR")
```

Arguments

`user`	Logical: prioritize the user level config even if a config in the current working directory is available.
`pkg`	Character: name of the package.

Value

Character: path to the config file.

Examples

```
omnipath_config_path()
```

`omnipath_for_cosmos` *OmniPath PPI for the COSMOS PKN*

Description

OmniPath PPI for the COSMOS PKN

Usage

```
omnipath_for_cosmos(
  organism = 9606L,
  resources = NULL,
  datasets = NULL,
  interaction_types = NULL,
  id_types = c("uniprot", "genesymbol"),
  ...
)
```

Arguments

`organism`	Character or integer: name or NCBI Taxonomy ID of the organism.
`resources`	Character: names of one or more resources. Correct spelling is important.
`datasets`	Character: one or more network datasets in OmniPath.
`interaction_types`	Character: one or more interaction type
`id_types`	Character: translate the protein identifiers to these ID types. Each ID type results two extra columns in the output, for the "source" and "target" sides of the interaction, respectively. The default ID type for proteins is Esembl Gene ID, and by default UniProt IDs and Gene Symbols are included. The UniProt IDs returned by the web service are left intact, while the Gene Symbols are queried from Ensembl. These Gene Symbols are different from the ones returned from the web service, and match the Ensembl Gene Symbols used by other components of the COSMOS PKN.
`...`	Further parameters to `omnipath_interactions`.

Value

Data frame with the columns source, target and sign.

See Also

- `cosmos_pkn`
- `omnipath-interactions`

Examples

```
op_cosmos <- omnipath_for_cosmos()
op_cosmos
```

`omnipath_load_config` *Load the package configuration from a config file*

Description

Load the package configuration from a config file

Load the coniguration of a certain package

Usage

```
omnipath_load_config(path = NULL, title = "default", user = FALSE, ...)

load_config(
  path = NULL,
  title = "default",
  user = FALSE,
  pkg = "OmnipathR",
  ...
)
```

Arguments

`path`	Path to the config file.
`title`	Load the config under this title. One config file might contain multple configurations, each identified by a title. If the title is not available the first section of the config file will be used.
`user`	Force to use the user level config even if a config file exists in the current directory. By default, the local config files have prioroty over the user level config.
`...`	Passed to `yaml::yaml.load_file`.
`pkg`	Character: name of the package

Value

Invisibly returns the config as a list.

Examples

```
## Not run:
# load the config from a custom config file:
omnipath_load_config(path = 'my_custom_omnipath_config.yml')

## End(Not run)
```

`omnipath_log` *Browse the current OmnipathR log file*

Description

Browse the current OmnipathR log file

Browse the latest log from a package

Usage

```
omnipath_log()

read_log(pkg = "OmnipathR")
```

Arguments

`pkg`	Character: name of a package.

Value

Returns ‘NULL‘.

See Also

`omnipath_logfile`

Examples

```
## Not run:
omnipath_log()
# then you can browse the log file, and exit with `q`

## End(Not run)
```

`omnipath_logfile` *Path to the current OmnipathR log file*

Description

Path to the current OmnipathR log file

Path to the current logfile of a package

Usage

```
omnipath_logfile()

logfile(pkg = "OmnipathR")
```

Arguments

pkg	Character: name of a package.

Value

Character: path to the current logfile, or `NULL` if no logfile is available.

See Also

`omnipath_log`

Examples

```
omnipath_logfile()
# [1] "/home/denes/omnipathr/omnipathr-log/omnipathr-20210309-1642.log"
```

`omnipath_msg` *Dispatch a message to the OmnipathR logger*

Description

Any package or script can easily send log messages and establish a logging facility with the fantastic ‘logger‘ package. This function serves the only purpose if you want to inject messages into the logger of OmnipathR. Otherwise we recommend to use the ‘logger‘ package directly.

Usage

```
omnipath_msg(level, ...)
```

Arguments

`level`	Character, numeric or class loglevel. A log level, if character one of the followings: "fatal", "error", "warn", "success", "info", "trace".
`...`	Arguments for string formatting, passed `sprintf` or `str_glue`.

Value

Returns ‘NULL‘.

Examples

```
omnipath_msg(
    level = 'success',
    'Talking to you in the name of OmnipathR, my favourite number is %d',
    round(runif(1, 1, 10))
)
```

`omnipath_query` *Download data from the OmniPath web service*

Description

This is the most generic method for accessing data from the OmniPath web service. All other functions retrieving data from OmniPath call this function with various parameters. In general, every query can retrieve data in tabular or JSON format, the tabular (data frame) being the default.

Usage

```
omnipath_query(
  query_type,
  organism = 9606L,
  resources = NULL,
  datasets = NULL,
  types = NULL,
  genesymbols = "yes",
```

```
  fields = NULL,
  default_fields = TRUE,
  silent = FALSE,
  logicals = NULL,
  download_args = list(),
  format = "data.frame",
  references_by_resource = TRUE,
  add_counts = TRUE,
  license = NULL,
  password = NULL,
  exclude = NULL,
  json_param = list(),
  strict_evidences = FALSE,
  genesymbol_resource = "UniProt",
  cache = NULL,
  ...
)
```

Arguments

`query_type`	Character: "interactions", "enzsub", "complexes", "annotations", or "intercell".
`organism`	Character or integer: name or NCBI Taxonomy ID of the organism. OmniPath is built of human data, and the web service provides orthology translated interactions and enzyme-substrate relationships for mouse and rat. For other organisms and query types, orthology translation will be called automatically on the downloaded human data before returning the result.
`resources`	Character vector: name of one or more resources. Restrict the data to these resources. For a complete list of available resources, call the ‘<query_type>_resources‘ functions for the query type of interst.
`datasets`	Character vector: name of one or more datasets. In the interactions query type a number of datasets are available. The default is caled "omnipath", and corresponds to the curated causal signaling network published in the 2016 OmniPath paper.
`types`	Character vector: one or more interaction types, such as "transcriptional" or "post_translational". For a full list of interaction types see ‘query_info("interaction")$types‘.
`genesymbols`	Character or logical: TRUE or FALS or "yes" or "no". Include the ‘genesymbols‘ column in the results. OmniPath uses UniProt IDs as the primary identifiers, gene symbols are optional.
`fields`	Character vector: additional fields to include in the result. For a list of available fields, call ‘query_info("interactions")‘.
`default_fields`	Logical: if TRUE, the default fields will be included.
`silent`	Logical: if TRUE, no messages will be printed. By default a summary message is printed upon successful download.
`logicals`	Character vector: fields to be cast to logical.
`download_args`	List: parameters to pass to the download function, which is ‘readr::read_tsv‘ by default, and ‘jsonlite::safe_load‘.
`format`	Character: if "json", JSON will be retrieved and processed into a nested list; any other value will return data frame.

`references_by_resource`	Logical: if TRUE,, in the ‘references‘ column the PubMed IDs will be prefixed with the names of the resources they are coming from. If FALSE, the ‘references‘ column will be a list of unique PubMed IDs.
`add_counts`	Logical: if TRUE, the number of references and number of resources for each record will be added to the result.
`license`	Character: license restrictions. By default, data from resources allowing "academic" use is returned by OmniPath. If you use the data for work in a company, you can provide "commercial" or "for-profit", which will restrict the data to those records which are supported by resources that allow for-profit use.
`password`	Character: password for the OmniPath web service. You can provide a special password here which enables the use of ‘license = "ignore"‘ option, completely bypassing the license filter.
`exclude`	Character vector: resource or dataset names to be excluded. The data will be filtered after download to remove records of the excluded datasets and resources.
`json_param`	List: parameters to pass to the ‘jsonlite::fromJSON‘ when processing JSON columns embedded in the downloaded data. Such columns are "extra_attrs" and "evidences". These are optional columns which provide a lot of extra details about interactions.
`strict_evidences`	Logical: reconstruct the "sources" and "references" columns of interaction data frames based on the "evidences" column, strictly filtering them to the queried datasets and resources. Without this, the "sources" and "references" fields for each record might contain information for datasets and resources other than the queried ones, because the downloaded records are a result of a simple filtering of an already integrated data frame.
`genesymbol_resource`	Character: "uniprot" (default) or "ensembl". The OmniPath web service uses the primary gene symbols as provided by UniProt. By passing "ensembl" here, the UniProt gene symbols will be replaced by the ones used in Ensembl. This translation results in a loss of a few records, and multiplication of another few records due to ambiguous translation.
`cache`	Logical: use caching, load data from and save to the. The cache directory by default belongs to the user, located in the user's default cache directory, and named "OmnipathR". Find out about it by `getOption("omnipathr.cachedir")`. Can be changed by `omnipath_set_cachedir`.
`...`	Additional parameters for the OmniPath web service. These parameters will be processed, validated and included in the query string. Many parameters are already explicitly set by the arguments above. A number of query type specific parameters are also available, learn more about these by the `query_info` function. For functions more specific than `omnipath_query`, arguments for all downstream functions are also passed here.

Value

Data frame (tibble) or list: the data returned by the OmniPath web service (or loaded from cache), after processing. Nested list if the "format" parameter is "json", otherwise a tibble.

Examples

```
interaction_data <- omnipath_query("interaction", datasets = "omnipath")
```

`interaction_data`

`omnipath_save_config` *Save the current package configuration*

Description

Save the current package configuration

Save the configuration of a certain package

Usage

```
omnipath_save_config(path = NULL, title = "default", local = FALSE)

save_config(path = NULL, title = "default", local = FALSE, pkg = "OmnipathR")
```

Arguments

`path`	Path to the config file. Directories and the file will be created if don't exist.
`title`	Save the config under this title. One config file might contain multiple configurations, each identified by a title.
`local`	Save into a config file in the current directory instead of a user level config file. When loading, the config in the current directory has priority over the user level config.
`pkg`	Character: name of the package

Value

Returns ‘NULL‘.

Examples

```
## Not run:
# after this, all downloads will default to commercial licenses
# i.e. the resources that allow only academic use will be excluded:
options(omnipathr.license = 'commercial')
omnipath_save_config()

## End(Not run)
```

`omnipath_set_cachedir` *Change the cache directory*

Description

Change the cache directory

Usage

```
omnipath_set_cachedir(path = NULL)
```

Arguments

`path`	Character: path to the new cache directory. If don't exist, the directories will be created. If the path is an existing cache directory, the package's cache database for the current session will be loaded from the database in the directory. If `NULL`, the cache directory will be set to its default path.

Value

Returns `NULL`.

Examples

```
tmp_cache <- tempdir()
omnipath_set_cachedir(tmp_cache)
# restore the default cache directory:
omnipath_set_cachedir()
```

`omnipath_set_console_loglevel` *Sets the log level for the console*

Description

Use this method to change during a session which messages you want to be printed on the console. Before loading the package, you can set it also by the config file, with the omnipathr.console_loglevel key.

Usage

```
omnipath_set_console_loglevel(level)
```

Arguments

`level`	Character or class 'loglevel'. The desired log level.

Value

Returns 'NULL'.

See Also

omnipath_set_logfile_loglevel

Examples

```
omnipath_set_console_loglevel('warn')
# or:
omnipath_set_console_loglevel(logger::WARN)
```

---

omnipath_set_logfile_loglevel

Sets the log level for the logfile

Description

Use this method to change during a session which messages you want to be written into the logfile. Before loading the package, you can set it also by the config file, with the "omnipathr.loglevel" key.

Usage

```
omnipath_set_logfile_loglevel(level)
```

Arguments

`level`	Character or class ‘loglevel‘. The desired log level.

Value

Returns ‘NULL‘.

See Also

omnipath_set_console_loglevel

Examples

```
omnipath_set_logfile_loglevel('info')
# or:
omnipath_set_logfile_loglevel(logger::INFO)
```

`omnipath_set_loglevel` *Sets the log level for the package logger*

Description

Sets the log level for the package logger

Sets the log level for any package

Usage

```
omnipath_set_loglevel(level, target = "logfile")

set_loglevel(level, target = "logfile", pkg = "OmnipathR")
```

Arguments

`level`	Character or class ‘loglevel‘. The desired log level.
`target`	Character, either ’logfile’ or ’console’
`pkg`	Character: name of the package.

Value

Returns ‘NULL‘.

Examples

```
omnipath_set_loglevel(logger::FATAL, target = 'console')
```

`omnipath_show_db` *Built in database definitions*

Description

Databases are resources which might be costly to load but can be used many times by functions which usually automatically load and retrieve them from the database manager. Each database has a lifetime and will be unloaded automatically upon expiry.

Usage

```
omnipath_show_db()
```

Value

A data frame with the built in database definitions.

Examples

```
database_definitions <- omnipath_show_db()
database_definitions
# # A tibble: 14 x 10
#    name       last_used           lifetime package  loader    loader_p.
#    <chr>      <dttm>                 <dbl> <chr>    <chr>     <list>
#  1 Gene Onto. 2021-04-04 20:19:15      300 Omnipat. go_ontol. <named l.
#  2 Gene Onto. NA                       300 Omnipat. go_ontol. <named l.
#  3 Gene Onto. NA                       300 Omnipat. go_ontol. <named l.
#  4 Gene Onto. NA                       300 Omnipat. go_ontol. <named l.
#  5 Gene Onto. NA                       300 Omnipat. go_ontol. <named l.
# ... (truncated)
# # . with 4 more variables: latest_param <list>, loaded <lgl>, db <list>,
# #   key <chr>
```

`omnipath_unlock_cache_db` *Removes the lock file from the cache directory*

Description

A lock file in the cache directory avoids simulatneous write and read. It's supposed to be removed after each read and write operation. This might not happen if the process crashes during such an operation. In this case you can manually call this function.

Usage

```
omnipath_unlock_cache_db()
```

Value

Logical: returns TRUE if the cache was locked and now is unlocked; FALSE if it was not locked.

Examples

```
omnipath_unlock_cache_db()
```

`only_from` *Recreate interaction data frame based on certain datasets and resources*

Description

Recreate interaction data frame based on certain datasets and resources

Usage

```
only_from(
  data,
  datasets = NULL,
  resources = NULL,
  exclude = NULL,
  .keep = FALSE
)
```

Arguments

`data`	An interaction data frame from the OmniPath web service with evidences column.
`datasets`	Character: a vector of dataset labels. Only evidences from these datasets will be used.
`resources`	Character: a vector of resource labels. Only evidences from these resources will be used.
`exclude`	Character vector of resource names to be excluded.
`.keep`	Logical: keep the "evidences" column.

Details

The OmniPath interactions database fully integrates all attributes from all resources for each interaction. This comes with the advantage that interaction data frames are ready for use in most of the applications; however, it makes it impossible to know which of the resources and references support the direction or effect sign of the interaction. This information can be recovered from the "evidences" column. The "evidences" column preserves all the details about interaction provenances. In cases when you want to use a faithful copy of a certain resource or dataset, this function will help you do so. Still, in most of the applications the best is to use the interaction data as it is returned by the web service.

Note: This function is automatically applied if the ‘strict_evidences‘ argument is passed to any function querying interactions (e.g. `omnipath-interactions`).

Value

A copy of the interaction data frame restricted to the given datasets and resources.

See Also

- `omnipath-interactions`
- `filter_evidences`
- `unnest_evidences`
- `from_evidences`

Examples

```
## Not run:
ci <- collectri(evidences = TRUE)
ci <- only_from(ci, datasets = 'collectri')

## End(Not run)
```

`ontology_ensure_id` *Only ontology IDs*

Description

Converts a mixture of ontology IDs and names to only IDs. If an element of the input is missing from the chosen ontology it will be dropped. This can happen if the ontology is a subset (slim) version, but also if the input is not a valid ID or name.

Usage

```
ontology_ensure_id(terms, db_key = "go_basic")
```

Arguments

`terms`	Character: ontology IDs or term names.
`db_key`	Character: key to identify the ontology database. For the available keys see `omnipath_show_db`.

Value

Character vector of ontology IDs.

Examples

```
ontology_ensure_id(c('mitochondrion inheritance', 'GO:0001754'))
# [1] "GO:0000001" "GO:0001754"
```

`ontology_ensure_name` *Only ontology term names*

Description

Converts a mixture of ontology IDs and names to only names. If an element of the input is missing from the chosen ontology it will be dropped. This can happen if the ontology is a subset (slim) version, but also if the input is not a valid ID or name.

Usage

```
ontology_ensure_name(terms, db_key = "go_basic")
```

Arguments

`terms`	Character: ontology IDs or term names.
`db_key`	Character: key to identify the ontology database. For the available keys see `omnipath_show_db`.

Value

Character vector of ontology term names.

Examples

```
ontology_ensure_name(c('reproduction', 'GO:0001754', 'foo bar'))
# [1] "eye photoreceptor cell differentiation" "reproduction"
```

`ontology_name_id` *Translate between ontology IDs and names*

Description

Makes sure that the output contains only valid IDs or term names. The input can be a mixture of IDs and names. The order of the input won't be preserved in the output.

Usage

```
ontology_name_id(terms, ids = TRUE, db_key = "go_basic")
```

Arguments

`terms`	Character: ontology IDs or term names.
`ids`	Logical: the output should contain IDs or term names.
`db_key`	Character: key to identify the ontology database. For the available keys see `omnipath_show_db`.

Value

Character vector of ontology IDs or term names.

Examples

```
ontology_name_id(c('mitochondrion inheritance', 'reproduction'))
# [1] "GO:0000001" "GO:0000003"
ontology_name_id(c('GO:0000001', 'reproduction'), ids = FALSE)
# [1] "mitochondrion inheritance" "reproduction"
```

---

organism_for *Make sure the resource supports the organism and it has the ID*

Description

Make sure the resource supports the organism and it has the ID

Usage

```
organism_for(organism, resource, error = TRUE)
```

Arguments

`organism`	Character or integer: name or NCBI Taxonomy ID of the organism.
`resource`	Charater: name of the resource.
`error`	Logical: raise an error if the organism is not supported in the resource. Otherwise it only emits a warning.

Value

Character: the ID of the organism as it is used by the resource. NA if the organism can not be translated to the required identifier type.

Examples

```
organism_for(10116, 'chalmers-gem')
# [1] "Rat"
organism_for(6239, 'chalmers-gem')
# [1] "Worm"
# organism_for('foobar', 'chalmers-gem')
# Error in organism_for("foobar", "chalmers-gem") :
# Organism `foobar` (common_name: `NA`; common_name: `NA`)
# is not supported by resource `chalmers-gem`. Supported organisms:
# Human, Mouse, Rat, Zebrafish, Drosophila melanogaster (Fruit fly),
# Caenorhabditis elegans (PRJNA13758).
```

---

orthology_translate_column *Translate a column of identifiers by orthologous gene pairs*

Description

Translate a column of identifiers by orthologous gene pairs

Usage

```
orthology_translate_column(
  data,
  column,
  id_type = NULL,
  target_organism = "mouse",
  source_organism = "human",
  resource = "oma",
  replace = FALSE,
  one_to_many = NULL,
  keep_untranslated = FALSE,
  translate_complexes = FALSE,
  uniprot_by_id_type = "entrez"
)
```

Arguments

`data`	A data frame with the column to be translated.
`column`	Name of a character column with identifiers of the source organism of type ‘id_type‘.
`id_type`	Type of identifiers in ‘column‘. Available ID types include "uniprot", "entrez", "ensg", "refseq" and "swissprot" for OMA, and "uniprot", "entrez", "genesymbol", "refseq" and "gi" for NCBI HomoloGene. If you want to translate an ID type not directly available in your preferred resource, use first `translate_ids` to translate to an ID type directly available in the orthology resource. If not provided, it is assumed the column name is the ID type.
`target_organism`	Name or NCBI Taxonomy ID of the target organism.
`source_organism`	Name or NCBI Taxonomy ID of the source organism.
`resource`	Character: source of the orthology mapping. Currently Orthologous Matrix (OMA) and NCBI HomoloGene are available, refer to them by "oma" and "homologene", respectively.
`replace`	Logical or character: replace the column with the translated identifiers, or create a new column. If it is character, it will be used as the name of the new column.
`one_to_many`	Integer: maximum number of orthologous pairs for one gene of the source organism. Genes mapping to higher number of orthologues will be dropped.
`keep_untranslated`	Logical: keep records without orthologous pairs. If ‘replace‘ is TRUE, this option is ignored, and untranslated records will be dropped. Genes with more than ‘one_to_many‘ orthologues will always be dropped.
`translate_complexes`	Logical: translate the complexes by translating their components.
`uniprot_by_id_type`	Character: translate NCBI HomoloGene to UniProt by this ID type. One of "genesymbol", "entrez", "refseq" or "gi".

Value

The data frame with identifiers translated to other organism.

`pathwaycommons_download` *Interactions from PathwayCommons*

Description

PathwayCommons (`http://www.pathwaycommons.org/`) provides molecular interactions from a number of databases, in either BioPAX or SIF (simple interaction format). This function retrieves all interactions in SIF format. The data is limited to the interacting pair and the type of the interaction.

Usage

```
pathwaycommons_download()
```

Value

A data frame (tibble) with interactions.

Examples

```
pc_interactions <- pathwaycommons_download()
 pc_interactions
# # A tibble: 1,884,849 x 3
#    from  type                        to
#    <chr> <chr>                       <chr>
#  1 A1BG  controls-expression-of      A2M
#  2 A1BG  interacts-with              ABCC6
#  3 A1BG  interacts-with              ACE2
#  4 A1BG  interacts-with              ADAM10
#  5 A1BG  interacts-with              ADAM17
# # . with 1,884,839 more rows
```

`pivot_annotations` *Converts annotation tables to a wide format*

Description

Use this method to reconstitute the annotation tables into the format of the original resources. With the ‘wide=TRUE‘ option `annotations` applies this function to the downloaded data.

Usage

```
pivot_annotations(annotations)
```

Arguments

`annotations`	A data frame of annotations downloaded from the OmniPath web service by `annotations`.

Value

A wide format data frame (tibble) if the provided data contains annotations from one resource, otherwise a list of wide format tibbles.

See Also

`annotations`

Examples

```
# single resource: the result is a data frame
disgenet <- annotations(resources = "DisGeNet")
disgenet <- pivot_annotations(disgenet)
disgenet
# # A tibble: 126,588 × 11
#    uniprot genesymbol entity_type disease      type  score   dsi   dpi
#    <chr>   <chr>      <chr>       <chr>        <chr> <dbl> <dbl> <dbl>
#  1 P04217  A1BG       protein     Schizophren. dise.  0.3  0.7   0.538
#  2 P04217  A1BG       protein     Hepatomegaly phen.  0.3  0.7   0.538
#  3 P01023  A2M        protein     Fibrosis, L. dise.  0.3  0.529 0.769
#  4 P01023  A2M        protein     Acute kidne. dise.  0.3  0.529 0.769
#  5 P01023  A2M        protein     Mental Depr. dise.  0.3  0.529 0.769
# # . with 126,583 more rows, and 3 more variables: nof_pmids <dbl>,
# #   nof_snps <dbl>, source <chr>

# multiple resources: the result is a list
annot_long <- annotations(
    resources = c("DisGeNet", "SignaLink_function", "DGIdb", "kinase.com")
)
annot_wide <- pivot_annotations(annot_long)
names(annot_wide)
# [1] "DGIdb"              "DisGeNet"           "kinase.com"
# [4] "SignaLink_function"
annot_wide$kinase.com
# # A tibble: 825 x 6
#    uniprot genesymbol entity_type group family subfamily
#    <chr>   <chr>      <chr>       <chr> <chr>  <chr>
#  1 P31749  AKT1       protein     AGC   Akt    NA
#  2 P31751  AKT2       protein     AGC   Akt    NA
#  3 Q9Y243  AKT3       protein     AGC   Akt    NA
#  4 O14578  CIT        protein     AGC   DMPK   CRIK
#  5 Q09013  DMPK       protein     AGC   DMPK   GEK
# # . with 815 more rows
```

`preppi_download` *Interactions from PrePPI*

Description

Retrieves predicted protein-protein interactions from the PrePPI database (`http://honig.c2b2.columbia.edu/preppi`). The interactions in this table are supposed to be correct with a > 0.5 probability.

Usage

```
preppi_download(...)
```

Arguments

`...`	Minimum values for the scores. The available scores are: str, protpep, str_max, red, ort, phy, coexp, go, total, exp and final. Furthermore, an operator can be passed, either `.op = '&'` or `.op = '|'`, which is then used for combined filtering by multiple scores.

Details

PrePPI is a combination of many prediction methods, each resulting a score. For an explanation of the scores see `https://honiglab.c2b2.columbia.edu/hfpd/help/Manual.html`. The minimum, median and maximum values of the scores:

Score	Minimum	Median	Maximum
str	0	5.5	6,495
protpep	0	3.53	38,138
str_max	0	17.9	38,138
red	0	1.25	24.4
ort	0	0	5,000
phy	0	2.42	2.42
coexp	0	2.77	45.3
go	0	5.86	181
total	0	1,292	106,197,000,000
exp	1	958	4,626
final	600	1,778	4.91e14

Value

A data frame (tibble) of interactions with scores, databases and literature references.

See Also

`preppi_filter`

Examples

```
preppi <- preppi_download()
preppi
# # A tibble: 1,545,710 x 15
#    prot1 prot2 str_score protpep_score str_max_score red_score ort_score
#    <chr> <chr>     <dbl>         <dbl>         <dbl>     <dbl>     <dbl>
#  1 Q131. P146.      18.6          6.45          18.6      4.25     0.615
#  2 P064. Q96N.       1.83        14.3           14.3      4.25     0
#  3 Q7Z6. Q8NC.       4.57         0              4.57     0         0
#  4 P370. P154.     485.           0            485.       1.77     0.615
#  5 O004. Q9NR.      34.0          0             34.0      0.512    0
# # . with 1,545,700 more rows, and 8 more variables: phy_score <dbl>,
# #   coexp_score <dbl>, go_score <dbl>, total_score <dbl>, dbs <chr>,
# #   pubs <chr>, exp_score <dbl>, final_score <dbl>
```

`preppi_filter` *Filter PrePPI interactions by scores*

Description

Filter PrePPI interactions by scores

Usage

```
preppi_filter(data, ..., .op = "&")
```

Arguments

`data`	A data frame of PrePPI interactions as provided by `preppi_download`.
`...`	Minimum values for the scores. The available scores are: str, protpep, str_max, red, ort, phy, coexp, go, total, exp and final. See more about the scores at `preppi_download`.
`.op`	The operator to combine the scores with: either `'&'` or `'|'`. With the former, only records where all scores are above the threshold will be kept; with the latter, records where at least one score is above its threshold will be kept.

Value

The input data frame (tibble) filtered by the score thresholds.

See Also

`preppi_download`

Examples

```
preppi <- preppi_download()
preppi_filtered <- preppi_filter(preppi, red = 10, str = 4.5, ort = 1)
nrow(preppi_filtered)
# [1] 8443
```

`print_bma_motif_es` *Prints BMA motifs to the screen from a sequence of edges*

Description

The motifs can be copy-pasted into a BMA canvas.

Usage

```
print_bma_motif_es(edge_seq, G, granularity = 2)
```

Arguments

`edge_seq`	An igraph edge sequence.
`G`	An igraph graph object.
`granularity`	Numeric: granularity value.

Value

Returns ‘NULL‘.

Examples

```
interactions <- omnipath(resources = "ARN")
graph <- interaction_graph(interactions)
print_bma_motif_es(igraph::E(graph)[1], graph)
# {"Model": {
#     "Name": "Omnipath motif",
#     "Variables":[{
#         "Name":"ULK1",
#         "Id":1,
#         "RangeFrom":0,
#         "RangeTo":2,
#         "Formula":""
#     },
#     {
#         "Name":"ATG13",
#         ...
#     }],
# ... (truncated)
# }}
```

`print_bma_motif_vs` *Prints BMA motifs to the screen from a sequence of nodes*

Description

The motifs can be copy-pasted into a BMA canvas.

Usage

```
print_bma_motif_vs(node_seq, G)
```

Arguments

`node_seq`	An igraph node sequence.
`G`	An igraph graph object.

Value

Returns ‘NULL‘.

Examples

```
interactions <- omnipath(resources = "ARN")
graph <- interaction_graph(interactions)
print_bma_motif_vs(
    igraph::all_shortest_paths(
        graph,
        from = 'ULK1',
        to = 'ATG13'
    )$res,
    graph
)
```

`print_interactions` *Print OmniPath interactions*

Description

Prints the interactions or enzyme-substrate relationships in a nice format.

Usage

```
print_interactions(interactions, refs = FALSE)
```

Arguments

`interactions`	Data frame with the interactions generated by any of the functions in `omnipath-interactions`.
`refs`	Logical: include PubMed IDs where available.

Value

Returns ‘NULL‘.

Examples

```
enzsub <- enzyme_substrate()
print_interactions(head(enzsub))
print_interactions(tail(enzsub), refs = TRUE)
print_interactions(
    dplyr::filter(
        enzsub,
        enzyme_genesymbol == 'MAP2K1',
        substrate_genesymbol == 'MAPK3'
    )
)

signor <- omnipath(resources = "SIGNOR")
print_interactions(head(signor))
#            source interaction            target n_resources
# 6 MAPK14 (Q16539)  ==( + )==> MAPKAPK2 (P49137)          23
# 4  TRPM7 (Q96QT4)  ==( + )==>    ANXA1 (P04083)          10
# 1  PRKG1 (Q13976)  ==( - )==>    TRPC3 (Q13507)           8
# 2  PTPN1 (P18031)  ==( - )==>    TRPV6 (Q9H1D0)           6
# 5 PRKACA (P17612)  ==( - )==>   MCOLN1 (Q9GZU1)           6
# 3  RACK1 (P63244)  ==( - )==>    TRPM6 (Q9BX84)           2
```

`print_path_es` *Prints network paths in an edge sequence*

Description

Pretty prints the interactions in a path.

Usage

```
print_path_es(edges, G)
```

Arguments

`edges`	An igraph edge sequence object.
`G`	igraph object (from ptms or any interaction dataset)

Value

Returns ‘NULL‘.

See Also

- `print_path_vs`

Examples

```
interactions <- omnipath(resources = "SignaLink3")
OPI_g <- interaction_graph(interactions = interactions)
print_path_es(
    suppressWarnings(igraph::shortest_paths(
        OPI_g,
        from = 'TYRO3',
        to = 'STAT3',
        output = 'epath'
    ))$epath[[1]],
    OPI_g
)
```

`print_path_vs` *Print networks paths given by node sequence*

Description

Prints the interactions in the path in a nice format.

Usage

```
print_path_vs(nodes, G)
```

Arguments

`nodes`	An igraph node sequence object.
`G`	An igraph graph object (from ptms or interactions)

Value

Returns ‘NULL‘.

See Also

`print_path_es`

Examples

```
interactions <- omnipath(resources = "SignaLink3")
OPI_g <- interaction_graph(interactions = interactions)
print_path_vs(
    igraph::all_shortest_paths(
        OPI_g,
        from = 'TYRO3',
        to = 'STAT3'
    )$vpath,
    OPI_g
)
enzsub <- enzyme_substrate(resources=c("PhosphoSite", "SIGNOR"))
enzsub_g <- enzsub_graph(enzsub)
print_path_vs(
    igraph::all_shortest_paths(
        enzsub_g,
        from = 'SRC',
        to = 'STAT1'
    )$res,
    enzsub_g
)
```

`pubmed_open` *Open one or more PubMed articles*

Description

Open one or more PubMed articles

Usage

```
pubmed_open(pmids, browser = NULL, sep = ";", max_pages = 25L)
```

Arguments

`pmids`	Character or numberic vector of one or more PubMed IDs.
`browser`	Character: name of the web browser executable. If ‘NULL‘, the default web browser will be used.
`sep`	Character: split the PubMed IDs by this separator.
`max_pages`	Numeric: largest number of pages to open. This is to prevent opening hundreds or thousands of pages at once.

Value

Returns ‘NULL‘.

Examples

```
interactions <- omnipath()
pubmed_open(interactions$references[1])
```

`query_info` *OmniPath query parameters*

Description

All parameter names and their possible values for a query type. Note: parameters with ‘NULL‘ values have too many possible values to list them.

Usage

```
query_info(query_type)
```

Arguments

`query_type`	Character: interactions, annotations, complexes, enz_sub or intercell.

Value

A named list with the parameter names and their possible values.

Examples

```
ia_param <- query_info('interactions')
ia_param$datasets[1:5]
# [1] "dorothea"    "kinaseextra" "ligrecextra" "lncrna_mrna" "mirnatarget"
```

`ramilowski_download` *Downloads ligand-receptor interactions from Ramilowski et al. 2015*

Description

Curated ligand-receptor pairs from Supplementary Table 2 of the article "A draft network of ligand-receptor mediated multicellular signaling in human" (`https://www.nature.com/articles/ncomms8866`).

Usage

```
ramilowski_download()
```

Value

A data frame (tibble) with interactions.

Examples

```
rami_interactions <- ramilowski_download()
rami_interactions
# # A tibble: 2,557 x 16
#    Pair.Name Ligand.Approved. Ligand.Name Receptor.Approv.
#    <chr>     <chr>            <chr>       <chr>
#  1 A2M_LRP1  A2M              alpha-2-ma. LRP1
#  2 AANAT_MT. AANAT            aralkylami. MTNR1A
#  3 AANAT_MT. AANAT            aralkylami. MTNR1B
#  4 ACE_AGTR2 ACE              angiotensi. AGTR2
#  5 ACE_BDKR. ACE              angiotensi. BDKRB2
# # . with 2,547 more rows, and 12 more variables: Receptor.Name <chr>,
# #   DLRP <chr>, HPMR <chr>, IUPHAR <chr>, HPRD <chr>,
# #   STRING.binding <chr>, STRING.experiment <chr>, HPMR.Ligand <chr>,
# #   HPMR.Receptor <chr>, PMID.Manual <chr>, Pair.Source <chr>,
# #   Pair.Evidence <chr>
```

`ramp_id_mapping_table` *Pairwise ID translation table from RaMP database*

Description

Pairwise ID translation table from RaMP database

Usage

```
ramp_id_mapping_table(from, to, version = "2.5.4")
```

Arguments

`from`	Character or Symbol. Name of an identifier type.
`to`	Character or Symbol. Name of an identifier type.
`version`	Character. The version of RaMP to download.

Value

Dataframe of pairs of identifiers.

See Also

- `ramp_sqlite`
- `ramp_tables`
- `ramp_table`
- `translate_ids`
- `id_types`
- `hmdb_table`
- `uniprot_full_id_mapping_table`
- `uniprot_id_mapping_table`
- `ensembl_id_mapping_table`
- `chalmers_gem_id_mapping_table`

Examples

```
ramp_id_mapping_table('hmdb', 'kegg')
```

`ramp_id_type` *RaMP identifier type label*

Description

RaMP identifier type label

Usage

```
ramp_id_type(label)
```

Arguments

`label`	Character: an ID type label, as shown in the table returned by `id_types`

Value

Character: the RaMP specific ID type label, or the input unchanged if it could not be translated (still might be a valid identifier name). These labels should be valid value names, as used in RaMP SQL database.

See Also

- `chalmers_gem_id_type`
- `uniprot_id_type`
- `ensembl_id_type`
- `uploadlists_id_type`

Examples

```
ramp_id_type("rhea")
# [1] "rhea-comp"
```

`ramp_sqlite` *Download and open RaMP database SQLite*

Description

Download and open RaMP database SQLite

Usage

```
ramp_sqlite(version = "2.5.4")
```

Arguments

`version`	Character. The version of RaMP to download.

Value

SQLite connection.

See Also

- `ramp_tables`

Examples

```
sqlite_con <- ramp_sqlite()
```

`ramp_table` *Return table from RaMP database*

Description

Return table from RaMP database

Usage

```
ramp_table(name, version = "2.5.4")
```

Arguments

`name`	Character. The name of the RaMP table to fetch.
`version`	Character. The version of RaMP to download.

Value

Character vector of table names in the RaMP SQLite database.

See Also

- `ramp_sqlite`
- `ramp_tables`

Examples

```
ramp_table('source')
```

`ramp_tables` *List tables in RaMP database*

Description

List tables in RaMP database

Usage

```
ramp_tables(version = "2.5.4")
```

Arguments

`version`	Character. The version of RaMP to download.

Value

Character vector of table names in the RaMP SQLite database.

See Also

- `ramp_sqlite`

Examples

```
ramp_tables()
```

`regnetwork_directions` *Transcription factor effects from RegNetwork*

Description

Transcription factor effects from RegNetwork

Usage

```
regnetwork_directions(organism = "human")
```

Arguments

`organism`	Character: either human or mouse.

Value

A data frame (tibble) of TF-target interactions with effect signs.

Examples

```
regn_dir <- regnetwork_directions()
regn_dir
# # A tibble: 3,954 x 5
#    source_genesymb. source_entrez target_genesymb. target_entrez
#    <chr>            <chr>         <chr>            <chr>
#  1 AHR              196           CDKN1B           1027
#  2 APLNR            187           PIK3C3           5289
#  3 APLNR            187           PIK3R4           30849
#  4 AR               367           KLK3             354
#  5 ARNT             405           ALDOA            226
# # . with 3,944 more rows, and 1 more variable: effect <dbl>
```

`regnetwork_download` *Interactions from RegNetwork*

Description

Downloads transcriptional and post-transcriptional regulatory interactions from the RegNetwork database (`http://www.regnetworkweb.org/`). The information about effect signs (stimulation or inhibition), provided by `regnetwork_directions` are included in the result.

Usage

```
regnetwork_download(organism = "human")
```

Arguments

`organism`	Character: either human or mouse.

Value

Data frame with interactions.

Examples

```
regn_interactions <- regnetwork_download()
regn_interactions
# # A tibble: 372,778 x 7
#    source_genesymb. source_entrez target_genesymb. target_entrez
#    <chr>            <chr>         <chr>            <chr>
#  1 USF1             7391          S100A6           6277
#  2 USF1             7391          DUSP1            1843
#  3 USF1             7391          C4A              720
#  4 USF1             7391          ABCA1            19
#  5 TP53             7157          TP73             7161
# # . with 372,768 more rows, and 3 more variables: effect <dbl>,
# #   source_type <chr>, target_type <chr>
```

---

relations_list_to_table

Table from a nested list of ontology relations

Description

Converting the nested list to a table is a more costly operation, it takes a few seconds. Best to do it only once, or pass `tables = TRUE` to `obo_parser`, and convert the data frame to list, if you also need it in list format.

Usage

```
relations_list_to_table(relations, direction = NULL)
```

Arguments

`relations`	A nested list of ontology relations (the "relations" element of the list returned by `obo_parser` in case its argument ‘tables‘ is `FALSE`).
`direction`	Override the direction (i.e. child -> parents or parent -> children). The nested lists produced by functions in the current package add an attribute "direction" thus no need to pass this value. If the attribute and the argument are both missing, the column will be named simply "side2" and it won’t be clear whether the relations point from "term" to "side2" or the other way around. The direction should be a character vector of length 2 with the values "parents" and "children".

Value

The relations converted to a data frame (tibble).

See Also

- `swap_relations`
- `relations_table_to_list`
- `obo_parser`

Examples

```
goslim_url <-
    "http://current.geneontology.org/ontology/subsets/goslim_generic.obo"
path <- tempfile()
httr::GET(goslim_url, httr::write_disk(path, overwrite = TRUE))
obo <- obo_parser(path, tables = FALSE)
unlink(path)
rel_tbl <- relations_list_to_table(obo$relations)
```

`relations_table_to_graph`

Graph from a table of ontology relations

Description

Graph from a table of ontology relations

Usage

```
relations_table_to_graph(relations)
```

Arguments

`relations`	A data frame of ontology relations (the "relations" element of the list returned by `obo_parser` in case its argument ‘tables‘ is `TRUE`).

Details

By default the relations point from child to parents, the edges in the graph will be of the same direction. Use `swap_relations` on the data frame to reverse the direction.

Value

The relations converted to an igraph graph object.

Examples

```
## Not run:
go <- get_db('go_basic')
go_graph <- relations_table_to_graph(go$relations)

## End(Not run)
```

`relations_table_to_list` *Nested list from a table of ontology relations*

Description

Nested list from a table of ontology relations

Usage

```
relations_table_to_list(relations)
```

Arguments

`relations`	A data frame of ontology relations (the "relations" element of the list returned by `obo_parser` in case its argument ‘tables‘ is `TRUE`).

Value

The relations converted to a nested list.

See Also

- `relations_list_to_table`
- `swap_relations`
- `obo_parser`

Examples

```
goslim_url <-
    "http://current.geneontology.org/ontology/subsets/goslim_generic.obo"
path <- tempfile()
httr::GET(goslim_url, httr::write_disk(path, overwrite = TRUE))
obo <- obo_parser(path, tables = TRUE)
unlink(path)
rel_list <- relations_table_to_list(obo$relations)
```

`remap_dorothea_download` *Downloads TF-target interactions from ReMap*

Description

ReMap (http://remap.univ-amu.fr/) is a database of ChIP-Seq experiments. It provides raw and merged peaks and CRMs (cis regulatory motifs) with their associations to regulators (TFs). TF-target relationships can be derived as it is written in Garcia-Alonso et al. 2019: "For ChIP-seq, we downloaded the binding peaks from ReMap and scored the interactions between each TF and each gene according to the distance between the TFBSs and the genes' transcription start sites. We evaluated different filtering strategies that consisted of selecting only the top-scoring 100, 200, 500, and 1000 target genes for each TF." (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6673718/#s1title). This function returns the top TF-target relationships as used in DoRothEA: https://github.com/saezlab/dorothea/blob/master/inst/scripts/02_chip_seq.R).

Usage

```
remap_dorothea_download()
```

Value

Data frame with TF-target relationships.

See Also

`remap_tf_target_download`

Examples

```
remap_interactions <- remap_dorothea_download()
remap_interactions
# # A tibble: 136,988 x 2
#    tf    target
#    <chr> <chr>
#  1 ADNP  ABCC1
#  2 ADNP  ABCC6
#  3 ADNP  ABHD5
#  4 ADNP  ABT1
#  5 ADNP  AC002066.1
# # . with 136,978 more rows
```

`remap_filtered` *Downloads TF-target interactions from ReMap*

Description

Downloads the ReMap TF-target interactions as processed by Garcia-Alonso et al. (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6673718/#s1title) and filters them based on a score threshold, the top targets and whether the TF is included in the TF census (Vaquerizas et al. 2009). The code for filtering is adapted from DoRothEA, written by Christian Holland.

Usage

```
remap_filtered(score = 100, top_targets = 500, only_known_tfs = TRUE)
```

Arguments

`score`	Numeric: a minimum score between 0 and 1000, records with lower scores will be excluded. If NULL no filtering performed.
`top_targets`	Numeric: the number of top scoring targets for each TF. Essentially the maximum number of targets per TF. If NULL the number of targets is not restricted.
`only_known_tfs`	Logical: whether to exclude TFs which are not in TF census.

Value

Data frame with TF-target relationships.

See Also

- `remap_tf_target_download`
- `remap_filtered`
- `tfcensus_download`

Examples

```
## Not run:
remap_interactions <- remap_filtered()
nrow(remap_interactions)
# [1] 145680

remap_interactions <- remap_filtered(top_targets = 100)
remap_interactions
# # A tibble: 30,330 x 2
#    source_genesymbol target_genesymbol
#    <chr>             <chr>
#   1 ADNP             ABCC1
#   2 ADNP             ABT1
#   3 ADNP             AC006076.1
#   4 ADNP             AC007792.1
#   5 ADNP             AC011288.2
# # . with 30,320 more rows

## End(Not run)
```

`remap_tf_target_download`

Downloads TF-target interactions from ReMap

Description

ReMap (http://remap.univ-amu.fr/) is a database of ChIP-Seq experiments. It provides raw and merged peaks and CRMs (cis regulatory motifs) with their associations to regulators (TFs). TF-target relationships can be derived as it is written in Garcia-Alonso et al. 2019: "For ChIP-seq, we downloaded the binding peaks from ReMap and scored the interactions between each TF and each gene according to the distance between the TFBSs and the genes' transcription start sites. We evaluated different filtering strategies that consisted of selecting only the top-scoring 100, 200, 500, and

1000 target genes for each TF." (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6673718/#s1title). This function retrieves the full processed TF-target list from the data deposited in https://zenodo.org/record/3713238.

Usage

```
remap_tf_target_download()
```

Value

Data frame with TF-target relationships.

See Also

- `remap_dorothea_download`
- `remap_filtered`

Examples

```
## Not run:
remap_interactions <- remap_tf_target_download()
remap_interactions
# # A tibble: 9,546,470 x 4
#    source_genesymbol target_genesymbol target_ensembl     score
#    <chr>             <chr>             <chr>              <dbl>
#  1 ADNP              PTPRS             ENSG00000105426.16  1000
#  2 AFF4              PRKCH             ENSG00000027075.14  1000
#  3 AHR               CTNND2            ENSG00000169862.18  1000
#  4 AR                PDE4D             ENSG00000113448.18  1000
#  5 ARID1A            PLEC              ENSG00000178209.14  1000
# # . with 9,546,460 more rows

## End(Not run)
```

`reset_config` *Restore the built-in default values of all config parameters of a package*

Description

Restore the built-in default values of all config parameters of a package

Restore the built-in default values of all config parameters of OmnipathR

Usage

```
reset_config(save = NULL, reset_all = FALSE, pkg = "OmnipathR")

omnipath_reset_config(...)
```

Arguments

`save`	If a path, the restored config will be also saved to this file. If TRUE, the config will be saved to the current default config path (see `omnipath_config_path`).
`reset_all`	Reset to their defaults also the options already set in the R options.
`pkg`	Character: name of a package
`...`	Ignored.

Value

The config as a list.

See Also

`omnipath_load_config`, `omnipath_save_config`

Examples

```
## Not run:
# restore the defaults and write them to the default config file:
omnipath_reset_config()
omnipath_save_config()

## End(Not run)
```

`resources` *Retrieve the available resources for a given query type*

Description

Collects the names of the resources available in OmniPath for a certain query type and optionally for a dataset within that.

Usage

```
resources(query_type, datasets = NULL, generic_categories = NULL)
```

Arguments

`query_type`	one of the query types ‘interactions‘, ‘enz_sub‘, ‘complexes‘, ‘annotations‘ or ‘intercell‘
`datasets`	currently within the ‘interactions‘ query type only, multiple datasets are available: ‘omnipath‘, ‘kinaseextra‘, ‘pathwayextra‘, ‘ligrecextra‘, ‘dorothea‘, ‘tf_target‘, ‘tf_mirna‘, ‘mirnatarget‘ and ‘lncrna_mrna‘.
`generic_categories`	for the ‘intercell‘ query type, restrict the search for some generic categories e.g. ‘ligand‘ or ‘receptor‘.

Value

a character vector with resource names

Examples

```
resources(query_type = "interactions")
```

`resources_colname` *Name of the column with the resources*

Description

Unfortunately the column title is different across the various query types in the OmniPath web service, so we need to guess.

Usage

```
resources_colname(data)
```

Arguments

`data`	A data frame downloaded by any `import_...` function in the current package.

Value

Character: the name of the column, if any of the column names matches.

Examples

```
co <- complexes()
resources_colname(co)
# [1] "sources"
```

`resources_in` *Collect resource names from a data frame*

Description

Collect resource names from a data frame

Usage

```
resources_in(data)
```

Arguments

`data`	A data frame from an OmniPath query.

Value

Character: resource names occuring in the data frame.

Examples

```
pathways <- omnipath_interactions()
resources_in(pathways)
```

`resource_info` *OmniPath resource information*

Description

The ‘resources‘ query type provides resource metadata in JSON format. Here we retrieve this JSON and return it as a nested list structure.

Usage

```
resource_info()
```

Value

A nested list structure with resource metadata.

Examples

```
resource_info()
```

`show_network` *Visualize node neighborhood with SigmaJS*

Description

This function takes an OmniPath interaction data frame as input and returns a sigmaJS object for the subgraph formed by the neighbors of a node of interest.

Usage

```
show_network(interactions, node = NULL)
```

Arguments

`interactions`	An OmniPath interaction data frame.
`node`	The node of interest.

Value

A sigmaJS object, check http://sigmajs.john-coene.com/index.html for further details and customization options.

Examples

```
## Not run:
# get interactions from omnipath
interactions <- omnipath()
# create and plot the network containing ATM neighbors
viz_sigmajs_neighborhood(interactions_df = interactions, int_node = "ATM")

## End(Not run)
```

`signed_ptms` *Causal effect enzyme-PTM interactions*

Description

Enzyme-substrate data does not contain sign (activation/inhibition), we generate this information based on the interaction network.

Usage

```
signed_ptms(
  enzsub = enzyme_substrate(),
  interactions = omnipath_interactions()
)
```

Arguments

`enzsub`	Enzyme-substrate data frame generated by `enzyme_substrate`
`interactions`	interaction data frame generated by an OmniPath interactions query: `omnipath-interactions`

Value

Data frame of enzyme-substrate relationships with is_inhibition and is_stimulation columns.

See Also

- `enzyme_substrate`
- `omnipath-interactions`

Examples

```
enzsub <- enzyme_substrate(resources = c("PhosphoSite", "SIGNOR"))
interactions <- omnipath_interactions()
enzsub <- signed_ptms(enzsub, interactions)
```

`simplify_intercell_network`

Simplify an intercell network

Description

The intercellular communication network data frames, created by `intercell_network`, are combinations of a network data frame with two copies of the intercell annotation data frames, all of them already having quite some columns. Here we keep only the names of the interacting pair, their intercellular communication roles, and the minimal information of the origin of both the interaction and the annotations. Optionally further columns can be selected.

Usage

```
simplify_intercell_network(network, ...)
```

Arguments

`network`	An intercell network data frame, as provided by `intercell_network`.
`...`	Optional, further columns to select.

Value

An intercell network data frame with some columns removed.

See Also

- `intercell_network`
- `filter_intercell_network`
- `unique_intercell_network`
- `intercell`
- `intercell_categories`
- `intercell_generic_categories`
- `intercell_summary`

Examples

```
icn <- intercell_network()
icn_s <- simplify_intercell_network(icn)
```

`static_table` *Retrieve a static table from OmniPath*

Description

A few resources and datasets are available also as plain TSV files and can be accessed without TLS. The purpose of these tables is to make the most often used OmniPath data available on computers with configuration issues. These tables are not the recommended way to access OmniPath data, and a warning is issued each time they are accessed.

Usage

```
static_table(
  query,
  resource,
  organism = 9606L,
  strict_evidences = TRUE,
  wide = TRUE,
  dorothea_levels = c("A", "B", "C")
)
```

Arguments

`query`	Character: a query type such as "annotations" or "interactions".
`resource`	Character: name of the resource or dataset, such as "CollecTRI" or "PROGENy".
`organism`	Integer: NCBI Taxonomy of the organism: 9606 for human, 10090 for mouse and 10116 for rat.
`strict_evidences`	Logical: restrict the evidences to the queried datasets and resources. If set to FALSE, the directions and effect signs and references might be based on other datasets and resources.
`wide`	Convert the annotation table to wide format, which corresponds more or less to the original resource. If the data comes from more than one resource a list of wide tables will be returned. See examples at `pivot_annotations`.
`dorothea_levels`	Vector detailing the confidence levels of the interactions to be downloaded. In dorothea, every TF-target interaction has a confidence score ranging from A to E, being A the most reliable interactions. By default here we take A, B and C level interactions (`c("A", "B", "C")`). It is to note that E interactions are not available in OmnipathR.

Value

A data frame (tibble) with the requested resource.

See Also

`static_tables`

Examples

```
static_table("annotations", "PROGENy")
```

`static_tables` *List the static tables available from OmniPath*

Description

A few resources and datasets are available also as plain TSV files and can be accessed without TLS. The purpose of these tables is to make the most often used OmniPath data available on computers with configuration issues. These tables are not the recommended way to access OmniPath data, and a warning is issued each time they are accessed.

Usage

```
static_tables()
```

Value

A data frame listing the available tables.

See Also

`static_table`

Examples

```
static_tables()
```

`stitch_actions` *Retrieve the STITCH actions dataset*

Description

Retrieve the STITCH actions dataset

Usage

```
stitch_actions(organism = "human", prefixes = FALSE)
```

Arguments

`organism`	Character or integer: name or NCBI Taxonomy ID of an organism. STITCH supports many organisms, please refer to their web site at `https://stitch.embl.de/`.
`prefixes`	Logical: include the prefixes in front of identifiers.

Value

Data frame of STITCH actions.

See Also

- stitch_actions
- stitch_links
- stitch_network

Examples

```
sta <- stitch_actions(organism = 'mouse')
```

`stitch_links` *Retrieve the STITCH links dataset*

Description

Retrieve the STITCH links dataset

Usage

```
stitch_links(organism = "human", prefixes = FALSE)
```

Arguments

`organism`	Character or integer: name or NCBI Taxonomy ID of an organism. STITCH supports many organisms, please refer to their web site at https://stitch.embl.de/.
`prefixes`	Logical: include the prefixes in front of identifiers.

Value

Data frame: organism specific STITCH links dataset.

See Also

- stitch_actions
- stitch_links
- stitch_network

Examples

```
stl <- stitch_links()
```

`stitch_network` *Chemical-protein interactions from STITCH*

Description

Chemical-protein interactions from STITCH

Usage

```
stitch_network(
  organism = "human",
  min_score = 700L,
  protein_ids = c("uniprot", "genesymbol"),
  metabolite_ids = c("hmdb", "kegg"),
  cosmos = FALSE
)
```

Arguments

`organism`	Character or integer: name or NCBI Taxonomy ID of an organism. STITCH supports many organisms, please refer to their web site at https://stitch.embl.de/.
`min_score`	Confidence cutoff used for STITCH connections (700 by default).
`protein_ids`	Character: translate the protein identifiers to these ID types. Each ID type results two extra columns in the output, for the "a" and "b" sides of the interaction, respectively. The default ID type for proteins is Esembl Protein ID, and by default UniProt IDs and Gene Symbols are included.
`metabolite_ids`	Character: translate the protein identifiers to these ID types. Each ID type results two extra columns in the output, for the "a" and "b" sides of the interaction, respectively. The default ID type for metabolites is PubChem CID, and HMDB IDs and KEGG IDs are included.
`cosmos`	Logical: use COSMOS format?

Value

A data frame of STITCH chemical-protein and protein-chemical interactions with their effect signs, and optionally with identifiers translated.

See Also

- `stitch_actions`
- `stitch_links`
- `stitch_remove_prefixes`

Examples

```
stn <- stitch_network(protein_ids = 'genesymbol', metabolite_ids = 'hmdb')
```

`stitch_remove_prefixes` *Remove the prefixes from STITCH identifiers*

Description

STITCH adds the NCBI Taxonomy ID as a prefix to Ensembl protein identifiers, e.g. "9606.ENSP00000170630", and "CID" followed by "s" or "m" (stereospecific or merged, respectively) in front of PubChem Compound Identifiers. It also pads the CID with zeros. This function removes these prefixes, leaving only the identifiers.

Usage

```
stitch_remove_prefixes(d, ..., remove = TRUE)
```

Arguments

`d`	Data frame, typically the output of `stitch_links` or `stitch_actions`.
`...`	Names of columns to remove prefixes from. NSE is supported.
`remove`	Logical: remove the prefixes? If FALSE, this function does nothing.

Value

Data frame with prefixes removed in the specified columns.

See Also

- `stitch_actions`
- `stitch_links`
- `stitch_network`

Examples

```
stitch_remove_prefixes(
    data.frame(a = c('9606.ENSP00000170630', 'CIDs00012345')),
    a
)
```

`subnetwork` *Extract a custom subnetwork from a large network*

Description

Extract a custom subnetwork from a large network

Usage

```
subnetwork(
  network,
  nodes = NULL,
  order = 1L,
  mode = "all",
  mindist = 0L,
  return_df = TRUE
)
```

Arguments

`network`	Either an OmniPath interaction data frame, or an igraph graph object.
`nodes`	Character or integer vector: names, identifiers or indices of the nodes to build the subnetwork around.
`order`	Integer: order of neighbourhood around nodes; i.e., number of steps starting from the provided nodes.
`mode`	Character: "all", "out" or "in". Follow directed edges from the provided nodes in any, outbound or inbound direction, respectively.
`mindist`	Integer: The minimum distance to include the vertex in the result.
`return_df`	Logical: return an interaction data frame instead of an igraph object.

Value

A network data frame or an igraph object, depending on the “return_df“ parameter.

See Also

- `interaction_graph`
- `graph_interaction`
- `show_network`

`swap_relations` *Reverse the direction of ontology relations*

Description

Reverse the direction of ontology relations

Usage

```
swap_relations(relations)
```

Arguments

`relations`	The ‘relations‘ component of the data returned by `obo_parser` or any ‘...ontology_download‘ function such as `go_ontology_download`. Depending on the `tables` argument of those functions the ‘relations‘ can be a data frame or a nested list.

Value

Same type as the input, but the relations swapped: if in the input these pointed from each child to the parents, in the output they point from each parent to their children, and vice versa.

See Also

- `relations_list_to_table`
- `relations_table_to_list`
- `obo_parser`

Examples

```
goslim_url <-
    "http://current.geneontology.org/ontology/subsets/goslim_generic.obo"
path <- tempfile()
httr::GET(goslim_url, httr::write_disk(path, overwrite = TRUE))
obo <- obo_parser(path)
unlink(path)
rel_swapped <- swap_relations(obo$relations)
```

`swissprots_only` *Retain only SwissProt IDs*

Description

Retain only SwissProt IDs

Usage

```
swissprots_only(uniprots, organism = 9606)
```

Arguments

`uniprots`	Character vector of UniProt IDs.
`organism`	Character or integer: name or identifier of the organism.

Value

Character vector with only SwissProt IDs.

Examples

```
swissprots_only(c("Q05BL1", "A0A654IBU3", "P00533"))
# [1] "P00533"
```

`tfcensus_download` *Downloads the list of transcription factors from TF census*

Description

Vaquerizas et al. published in 2009 a list of transcription factors. This function retrieves Supplementary Table 2 from the article (`http://www.nature.com/nrg/journal/v10/n4/index.html`).

Usage

```
tfcensus_download()
```

Value

A data frame (tibble) listing transcription factors.

Examples

```
tfcensus <- tfcensus_download()
tfcensus
# # A tibble: 1,987 x 7
#    Class `Ensembl ID` `IPI ID` `Interpro DBD` `Interpro DNA-b.
#    <chr> <chr>        <chr>    <chr>          <chr>
#  1 a     ENSG0000000. IPI0021. NA             IPR001289
#  2 a     ENSG0000000. IPI0004. IPR000047;IPR. NA
#  3 a     ENSG0000000. IPI0001. IPR001356;IPR. NA
#  4 a     ENSG0000000. IPI0029. IPR000910;IPR. NA
#  5 a     ENSG0000000. IPI0001. IPR007087;IPR. IPR006794
# # . with 1,977 more rows, and 2 more variables: `HGNC symbol` <chr>,
# # `Tissue-specificity` <chr>
```

`translate_ids` *Translate gene, protein and small molecule identifiers*

Description

Translates a vector of identifiers, resulting a new vector, or a column of identifiers in a data frame by creating another column with the target identifiers.

Usage

```
translate_ids(
  d,
  ...,
  uploadlists = FALSE,
  ensembl = FALSE,
  hmdb = FALSE,
  ramp = FALSE,
  chalmers = FALSE,
  entity_type = NULL,
```

```
  keep_untranslated = TRUE,
  return_df = FALSE,
  organism = 9606,
  reviewed = TRUE,
  complexes = NULL,
  complexes_one_to_many = NULL
)
```

Arguments

`d`	Character vector or data frame.
`...`	At least two arguments, with or without names. The first of these arguments describes the source identifier, the rest of them describe the target identifier(s). The values of all these arguments must be valid identifier types as shown in Details. The names of the arguments are column names. In case of the first (source) ID the column must exist. For the rest of the IDs new columns will be created with the desired names. For ID types provided as arguments without names, the name of the ID type will be used for column name.
`uploadlists`	Force using the `uploadlists` service from UniProt. By default the plain query interface is used (implemented in `uniprot_full_id_mapping_table` in this package). If any of the provided ID types is only available in the uploadlists service, it will be automatically selected. The plain query interface is preferred because in the long term, with caching, it requires less download and data storage.
`ensembl`	Logical: use data from Ensembl BioMart instead of UniProt.
`hmdb`	Logical: use HMDB ID translation data.
`ramp`	Logical: use RaMP ID translation data.
`chalmers`	Logical: use ID translation data from Chalmers Sysbio GEM.
`entity_type`	Character: "gene" and "smol" are short symbols for proteins, genes and small molecules respectively. Several other synonyms are also accepted.
`keep_untranslated`	In case the output is a data frame, keep the records where the source identifier could not be translated. At these records the target identifier will be NA.
`return_df`	Return a data frame even if the input is a vector.
`organism`	Character or integer, name or NCBI Taxonomy ID of the organism (by default 9606 for human). Matters only if `uploadlists` is `FALSE`.
`reviewed`	Translate only reviewed (`TRUE`), only unreviewed (`FALSE`) or both (`NULL`) UniProt records. Matters only if `uploadlists` is `FALSE`.
`complexes`	Logical: translate complexes by their members. Only complexes where all members can be translated will be included in the result. If `NULL`, the option `omnipathr.complex_translation` will be used.
`complexes_one_to_many`	Logical: allow combinatorial expansion or use only the first target identifier for each member of each complex. If `NULL`, the option `omnipathr.complex_translation_one_to_man` will be used.

Details

This function, depending on the `uploadlists` parameter, uses either the uploadlists service of UniProt or plain UniProt queries to obtain identifier translation tables. The possible values for `from` and `to` are the identifier type abbreviations used in the UniProt API, please refer to the table here: `https://www.uniprot.org/help/api_idmapping`. In addition, simple synonyms are available which realize a uniform API for the uploadlists and UniProt query based backends. These are the followings:

OmnipathR	**Uploadlists**	**UniProt query**	**Ensembl BioMart**
uniprot	ACC	id	uniprotswissprot
uniprot_entry	ID	entry name	
trembl	*reviewed = FALSE*	*reviewed = FALSE*	uniprotsptrembl
genesymbol	GENENAME	genes(PREFERRED)	external_gene_name
genesymbol_syn		genes(ALTERNATIVE)	external_synonym
hgnc	HGNC_ID	database(HGNC)	hgnc_symbol
entrez	P_ENTREZGENEID	database(GeneID)	
ensembl	ENSEMBL_ID		ensembl_gene_id
ensg	ENSEMBL_ID		ensembl_gene_id
enst	ENSEMBL_TRS_ID	database(Ensembl)	ensembl_transcript_id
ensp	ENSEMBL_PRO_ID		ensembl_peptide_id
ensgg	ENSEMBLGENOME_ID		
ensgt	ENSEMBLGENOME_TRS_ID		
ensgp	ENSEMBLGENOME_PRO_ID		
protein_name		protein names	
pir	PIR	database(PIR)	
ccds		database(CCDS)	
refseqp	P_REFSEQ_AC	database(refseq)	
ipro			interpro
ipro_desc			interpro_description
ipro_sdesc			interpro_short_description
wikigene			wikigene_name
rnacentral			rnacentral
gene_desc			description
wormbase		database(WormBase)	
flybase		database(FlyBase)	
xenbase		database(Xenbase)	
zfin		database(ZFIN)	
pbd	PBD_ID	database(PDB)	pbd

For a complete list of ID types and their synonyms, including metabolite and chemical ID types which are not shown here, see `id_types`.

The mapping between identifiers can be ambiguous. In this case one row in the original data frame yields multiple rows or elements in the returned data frame or vector(s).

Value

- Data frame: if the input is a data frame or the input is a vector and `return_df` is `TRUE`.
- Vector: if the input is a vector, there is only one target ID type and `return_df` is `FALSE`.
- List of vectors: if the input is a vector, there are more than one target ID types and `return_df` is `FALSE`. The names of the list will be ID types (as they were column names, see the description of the `...` argument), and the list will also include the source IDs.

See Also

- `translate_ids_multi`
- `uniprot_id_mapping_table`
- `uniprot_full_id_mapping_table`
- `ensembl_id_mapping_table`
- `hmdb_id_mapping_table`
- `id_types`
- `ensembl_id_type`
- `uniprot_id_type`
- `uploadlists_id_type`
- `hmdb_id_type`
- `chalmers_gem_id_type`

Examples

```
d <- data.frame(uniprot_id = c('P00533', 'Q9ULV1', 'P43897', 'Q9Y2P5'))
d <- translate_ids(d, uniprot_id = uniprot, genesymbol)
d
#   uniprot_id genesymbol
# 1     P00533       EGFR
# 2     Q9ULV1       FZD4
# 3     P43897       TSFM
# 4     Q9Y2P5    SLC27A5
```

`translate_ids_multi` *Translate gene, protein and small molecule identifiers from multiple columns*

Description

Especially when translating network interactions, where two ID columns exist (source and target), it is convenient to call the same ID translation on multiple columns. The `translate_ids` function is already able to translate to multiple ID types in one call, but is able to work only from one source column. Here too, multiple target IDs are supported. The source columns can be listed explicitely, or they might share a common stem, in this case the first element of ... will be used as stem, and the column names will be created by adding the `suffixes`. The `suffixes` are also used to name the target columns. If no `suffixes` are provided, the name of the source columns will be added to the name of the target columns. ID types can be defined the same way as for `translate_ids`. The only limitation is that, if the source columns are provided as stem+suffixes, they must be the same ID type.

Usage

```
translate_ids_multi(
  d,
  ...,
  suffixes = NULL,
```

```
  suffix_sep = "_",
  uploadlists = FALSE,
  ensembl = FALSE,
  hmdb = FALSE,
  chalmers = FALSE,
  entity_type = NULL,
  keep_untranslated = TRUE,
  organism = 9606,
  reviewed = TRUE
)
```

Arguments

`d`	A data frame.
`...`	At least two arguments, with or without names. These arguments describe identifier columns, either the ones we translate from (source), or the ones we translate to (target). Columns existing in the data frame will be used as source columns. All the rest will be considered target columns. Alternatively, the source columns can be defined as a stem and a vector of suffixes, plus a separator between the stem and suffix. In this case, the source columns will be the ones that exist in the data frame with the suffixes added. The values of all these arguments must be valid identifier types as shown at `translate_ids`. If ID type is provided only for the first source column, the rest of the source columns will be assumed to have the same ID type. For the target identifiers new columns will be created with the desired names, with the suffixes added. If no suffixes provided, the names of the source columns will be used instead.
`suffixes`	Column name suffixes in case the names should be composed of stem and suffix.
`suffix_sep`	Character: separator between the stem and suffixes.
`uploadlists`	Force using the ‘uploadlists‘ service from UniProt. By default the plain query interface is used (implemented in `uniprot_full_id_mapping_table` in this package). If any of the provided ID types is only available in the uploadlists service, it will be automatically selected. The plain query interface is preferred because in the long term, with caching, it requires less download and data storage.
`ensembl`	Logical: use data from Ensembl BioMart instead of UniProt.
`hmdb`	Logical: use HMDB ID translation data.
`chalmers`	Logical: use ID translation data from Chalmers Sysbio GEM.
`entity_type`	Character: "gene" and "smol" are short symbols for proteins, genes and small molecules respectively. Several other synonyms are also accepted.
`keep_untranslated`	In case the output is a data frame, keep the records where the source identifier could not be translated. At these records the target identifier will be NA.
`organism`	Character or integer, name or NCBI Taxonomy ID of the organism (by default 9606 for human). Matters only if `uploadlists` is `FALSE`.
`reviewed`	Translate only reviewed (`TRUE`), only unreviewed (`FALSE`) or both (`NULL`) UniProt records. Matters only if `uploadlists` is `FALSE`.

Value

A data frame with all source columns translated to all target identifiers. The number of new columns is the product of source and target columns. The target columns are distinguished by the suffexes added to their names.

See Also

`translate_ids`

Examples

```
ia <- omnipath()
translate_ids_multi(ia, source = uniprot, target, ensp, ensembl = TRUE)
```

`trembls_only` *Retain only TrEMBL IDs*

Description

Retain only TrEMBL IDs

Usage

```
trembls_only(uniprots, organism = 9606)
```

Arguments

`uniprots`	Character vector of UniProt IDs.
`organism`	Character or integer: name or identifier of the organism.

Value

Character vector with only TrEMBL IDs.

Examples

```
trembls_only(c("Q05BL1", "A0A654IBU3", "P00533"))
# [1] "Q05BL1" "A0A654IBU3"
```

`trrust_download` *Downloads TF-target interactions from TRRUST*

Description

TRRUST v2 (`https://www.grnpedia.org/trrust/`) is a database of literature mined TF-target interactions for human and mouse.

Usage

```
trrust_download(organism = "human")
```

Arguments

`organism`	Character: either "human" or "mouse".

Value

A data frame of TF-target interactions.

Examples

```
trrust_interactions <- trrust_download()
trrust_interactions
# # A tibble: 11,698 x 4
#    source_genesymbol target_genesymbol effect reference
#    <chr>             <chr>              <dbl> <chr>
#  1 AATF              BAX                   -1 22909821
#  2 AATF              CDKN1A                 0 17157788
#  3 AATF              KLK3                   0 23146908
#  4 AATF              MYC                    1 20549547
#  5 AATF              TP53                   0 17157788
#  6 ABL1              BAX                    1 11753601
#  7 ABL1              BCL2                  -1 11753601
# # . with 11,688 more rows
```

`uniprot_full_id_mapping_table` *Creates an ID translation table from UniProt data*

Description

Creates an ID translation table from UniProt data

Usage

```
uniprot_full_id_mapping_table(
  to,
  from = "accession",
  reviewed = TRUE,
  organism = 9606
)
```

Arguments

`to`	Character or symbol: target ID type. See Details for possible values.
`from`	Character or symbol: source ID type. See Details for possible values.
`reviewed`	Translate only reviewed (`TRUE`), only unreviewed (`FALSE`) or both (`NULL`) UniProt records.
`organism`	Integer, NCBI Taxonomy ID of the organism (by default 9606 for human).

Details

For both source and target ID type, this function accepts column codes used by UniProt and some simple shortcuts defined here. For the UniProt codes please refer to https://www.uniprot.org/help/uniprotkb The shortcuts are entrez, genesymbol, genesymbol_syn (synonym gene symbols), hgnc, embl, refseqp (RefSeq protein), enst (Ensembl transcript), uniprot_entry (UniProtKB AC, e.g. EGFR_HUMAN), protein_name (full name of the protein), uniprot (UniProtKB ID, e.g. P00533). For a complete table please refer to `translate_ids`.

Value

A data frame (tibble) with columns ‘From‘ and ‘To‘, UniProt IDs and the corresponding foreign IDs, respectively.

See Also

- `translate_ids`
- `ensembl_id_mapping_table`
- `uniprot_id_mapping_table`

Examples

```
uniprot_entrez <- uniprot_full_id_mapping_table(to = 'entrez')
uniprot_entrez
# # A tibble: 20,723 x 2
#    From   To
#    <chr>  <chr>
#  1 Q96R72 NA
#  2 Q9UKL2 23538
#  3 Q9H205 144125
#  4 Q8NGN2 219873
#  5 Q8NGC1 390439
# # . with 20,713 more rows
```

`uniprot_genesymbol_cleanup`

TrEMBL to SwissProt by gene names

Description

TrEMBL to SwissProt by gene names

Usage

```
uniprot_genesymbol_cleanup(uniprots, organism = 9606, only_trembls = TRUE)
```

Arguments

`uniprots`	Character vector possibly containing TrEMBL IDs.
`organism`	Character or integer: organism name or identifier.
`only_trembls`	Attempt to convert only known TrEMBL IDs of the organism. This is the recommended practice.

Details

Sometimes one gene or protein is represented by multiple identifiers in UniProt. These are typically slightly different isoforms, some of them having TrEMBL IDs, some of the SwissProt. For the purposes of most systems biology application, the most important is to identify the protein or gene in a way that we can recognize it in other datasets. Unfortunately UniProt or Ensembl do not seem to offer solution for this issue. Hence, if we find that a TrEMBL ID has a gene name which is also associated with a SwissProt ID, we replace this TrEMBL ID by that SwissProt. There might be a minor difference in their sequence, but most of the omics analyses do not even consider isoforms. And it is quite possible that later UniProt will convert the TrEMBL record to an isoform within the SwissProt record. Typically this translation is not so important (but still beneficial) for human, but for other organisms it is critical especially when translating from foreign identifiers.

This function accepts a mixed input of UniProt IDs and provides a distinct translation table that you can use to translate your data.

Value

Data frame with two columns: "input" and "output". The first one contains all identifiers from the input vector ‘uniprots‘. The second one has the corresponding identifiers which are either SwissProt IDs with gene names identical to the TrEMBL IDs in the input, or if no such records are available, the output has the input items unchanged.

Examples

```
## Not run:
uniprot_genesymbol_cleanup('Q6PB82', organism = 10090)
# # A tibble: 1 × 2
#   input  output
#   <chr>  <chr>
# 1 Q6PB82 O70405

## End(Not run)
```

`uniprot_idmapping_id_types` *ID types available in the UniProt ID Mapping service*

Description

ID types available in the UniProt ID Mapping service

Usage

```
uniprot_idmapping_id_types()
```

Value

A data frame listing the ID types.

Examples

```
uniprot_idmapping_id_types()
```

`uniprot_id_mapping_table` *ID translation data from UniProt ID Mapping*

Description

Retrieves an identifier translation table from the UniProt ID Mapping service (https://www.uniprot.org/help/id_mapping).

Usage

```
uniprot_id_mapping_table(identifiers, from, to, chunk_size = NULL)
```

Arguments

`identifiers`	Character vector of identifiers
`from`	Character or symbol: type of the identifiers provided. See Details for possible values.
`to`	Character or symbol: identifier type to be retrieved from UniProt. See Details for possible values.
`chunk_size`	Integer: query the identifiers in chunks of this size. If you are experiencing download failures, try lower values.

Details

This function uses the uploadlists service of UniProt to obtain identifier translation tables. The possible values for ‘from‘ and ‘to‘ are the identifier type abbreviations used in the UniProt API, please refer to the table here: `uniprot_idmapping_id_types` or the table of synonyms supported by the current package: `translate_ids`. Note: if the number of identifiers is larger than the chunk size the log message about the cache origin is not guaranteed to be correct (most of the times it is still correct).

Value

A data frame (tibble) with columns ‘From‘ and ‘To‘, the identifiers provided and the corresponding target IDs, respectively.

See Also

`translate_ids`

Examples

```
uniprot_genesymbol <- uniprot_id_mapping_table(
    c('P00533', 'P23771'), uniprot, genesymbol
)
uniprot_genesymbol
# # A tibble: 2 x 2
#   From   To
#   <chr>  <chr>
# 1 P00533 EGFR
# 2 P23771 GATA3
```

`uniprot_id_type` *UniProt identifier type label*

Description

UniProt identifier type label

Usage

```
uniprot_id_type(label)
```

Arguments

`label`	Character: an ID type label, as shown in the table at `translate_ids`

Value

Character: the UniProt specific ID type label, or the input unchanged if it could not be translated (still might be a valid identifier name). This is the label that one can use in UniProt REST queries.

See Also

- `ensembl_id_type`
- `uploadlists_id_type`

Examples

```
ensembl_id_type("entrez")
# [1] "database(GeneID)"
```

`unique_intercell_network`

Unique intercellular interactions

Description

In the intercellular network data frames produced by `intercell_network`, by default each pair of annotations for an interaction is represented in a separate row. This function drops the annotations and keeps only the distinct interacting pairs.

Usage

```
unique_intercell_network(network, ...)
```

Arguments

`network`	An intercellular network data frame as produced by `intercell_network`.
`...`	Additional columns to keep. Note: if these have multiple values for an interacting pair, only the first row will be preserved.

Value

A data frame with interacting pairs and interaction attributes.

See Also

- `intercell_network`
- `simplify_intercell_network`
- `filter_intercell_network`
- `intercell`
- `intercell_categories`
- `intercell_generic_categories`
- `intercell_summary`

Examples

```
icn <- intercell_network()
icn_unique <- unique_intercell_network(icn)
```

`unnest_evidences` *Separate evidences by direction and effect sign*

Description

Separate evidences by direction and effect sign

Usage

```
unnest_evidences(data, longer = FALSE, .keep = FALSE)
```

Arguments

`data`	An interaction data frame with "evidences" column.
`longer`	Logical: If TRUE, the "evidences" column is split into rows.
`.keep`	Logical: keep the "evidences" column. When unnesting to longer data frame, the "evidences" column will contain the unnested evidences, while the original column will be retained under the "all_evidences" name (if ‘.keep = TRUE‘).

Value

The data frame with new columns or new rows by direction and sign.

See Also

- `only_from`
- `filter_evidences`
- `from_evidences`

Examples

```
## Not run:
op <- omnipath_interactions(fields = "evidences")
op <- unnest_evidences(op)
colnames(op)

## End(Not run)
```

`uploadlists_id_type` *UniProt Uploadlists identifier type label*

Description

UniProt Uploadlists identifier type label

Usage

```
uploadlists_id_type(label, side = "from")
```

Arguments

`label`	Character: an ID type label, as shown in the table at `translate_ids`
`side`	Character: either "from" or "to": direction of the mapping.

Value

Character: the UniProt Uploadlists specific ID type label, or the input unchanged if it could not be translated (still might be a valid identifier name). This is the label that one can use in UniProt Uploadlists (ID Mapping) queries.

See Also

- `ensembl_id_type`
- `uniprot_id_type`
- `hmdb_id_type`
- `chalmers_gem_id_type`

Examples

```
ensembl_id_type("entrez")
# [1] "GeneID"
```

`vinayagam_download` *Protein-protein interactions from Vinayagam 2011*

Description

Retrieves the Supplementary Table S6 from Vinayagam et al. 2011. Find out more at `https://doi.org/10.1126/scisignal.2001699`.

Usage

```
vinayagam_download()
```

Value

A data frame (tibble) with interactions.

Examples

```
vinayagam_interactions <- vinayagam_download()
vinayagam_interactions
# # A tibble: 34,814 x 5
#    `Input-node Gen. `Input-node Gen. `Output-node Ge. `Output-node Ge.
#    <chr>                       <dbl> <chr>                       <dbl>
#  1 C1orf103                    55791 MNAT1                        4331
#  2 MAST2                       23139 DYNLL1                       8655
#  3 RAB22A                      57403 APPL2                       55198
#  4 TRAP1                       10131 EXT2                         2132
#  5 STAT2                        6773 COPS4                       51138
```

```
# # . with 34,804 more rows, and 1 more variable:
# # `Edge direction score` <dbl>
```

`walk_ontology_tree` *All nodes of a subtree starting from the selected nodes*

Description

Starting from the selected nodes, recursively walks the ontology tree until it reaches either the root or leaf nodes. Collects all visited nodes.

Usage

```
walk_ontology_tree(
  terms,
  ancestors = TRUE,
  db_key = "go_basic",
  ids = TRUE,
  method = "gra",
 relations = c("is_a", "part_of", "occurs_in", "regulates", "positively_regulates",
    "negatively_regulates")
)
```

Arguments

`terms`	Character vector of ontology term IDs or names. A mixture of IDs and names can be provided.
`ancestors`	Logical: if `FALSE` the ontology tree is traversed towards the leaf nodes; if `TRUE`, the tree is traversed until the root. The former returns the ancestors (parents), the latter the descendants (children).
`db_key`	Character: key to identify the ontology database. For the available keys see `omnipath_show_db`.
`ids`	Logical: whether to return IDs or term names.
`method`	Character: either "gra" or "lst". The implementation to use for traversing the ontology tree. The graph based implementation is faster than the list based, the latter will be removed in the future.
`relations`	Character vector of ontology relation types. Only these relations will be used.

Details

Note: this function relies on the database manager, the first call might take long because of the database load process. Subsequent calls within a short period should be faster. See `get_ontology_db`.

Value

Character vector of ontology IDs. If the input terms are all leaves or roots `NULL` is returned. The starting nodes won't be included in the result unless they fall onto the traversal path from other nodes.

See Also

- omnipath_show_db
- get_ontology_db

Examples

```
walk_ontology_tree(c('GO:0006241', 'GO:0044211'))
# [1] "GO:0006139" "GO:0006220" "GO:0006221" "GO:0006241" "GO:0006725"
# [6] "GO:0006753" "GO:0006793" "GO:0006796" "GO:0006807" "GO:0008150"
# ... (truncated)
walk_ontology_tree(c('GO:0006241', 'GO:0044211'), ancestors = FALSE)
# [1] "GO:0044210" "GO:0044211"
walk_ontology_tree(
    c('GO:0006241', 'GO:0044211'),
    ancestors = FALSE,
    ids = FALSE
)
# [1] "'de novo' CTP biosynthetic process" "CTP salvage"
```

`with_extra_attrs` *Interaction records having certain extra attributes*

Description

Interaction records having certain extra attributes

Usage

```
with_extra_attrs(data, ...)
```

Arguments

`data`	An interaction data frame.
`...`	The name(s) of the extra attributes; NSE is supported.

Value

The data frame filtered to the records having the extra attribute.

See Also

- extra_attrs
- has_extra_attrs
- extra_attrs_to_cols
- filter_extra_attrs
- extra_attr_values

Examples

```
i <- omnipath(fields = "extra_attrs")
with_extra_attrs(i, Macrophage_type)
```

`with_references` *Interactions having references*

Description

Interactions having references

Usage

```
with_references(data, resources = NULL)
```

Arguments

`data`	An interaction data frame.
`resources`	Character: consider only these resources. If ‘NULL‘, records with any reference will be accepted.

Value

A subset of the input interaction data frame.

Examples

```
cc <- import_post_translational_interactions(resources = 'CellChatDB')
with_references(cc, 'CellChatDB')
```

`zenodo_download` *Retrieves data from Zenodo*

Description

Zenodo is a repository of large scientific datasets. Many projects and publications make their datasets available at Zenodo. This function downloads an archive from Zenodo and extracts the requested file.

Usage

```
zenodo_download(
  path,
  reader = NULL,
  reader_param = list(),
  url_key = NULL,
  zenodo_record = NULL,
  zenodo_fname = NULL,
  url_param = list(),
  url_key_param = list(),
  ...
)
```

Arguments

`path`	Character: path to the file within the archive.
`reader`	Optional, a function to read the connection.
`reader_param`	List: arguments for the reader function.
`url_key`	Character: name of the option containing the URL
`zenodo_record`	The Zenodo record ID, either integer or character.
`zenodo_fname`	The file name within the record.
`url_param`	List: variables to insert into the URL string (which is returned from the options).
`url_key_param`	List: variables to insert into the ‘url_key‘.
`...`	Passed to `archive_extractor`

Value

A connection

Examples

```
# an example from the OmnipathR::remap_tf_target_download function:
remap_dorothea <- zenodo_download(
    zenodo_record = 3713238,
    zenodo_fname = 'tf_target_sources.zip',
    path = (
        'tf_target_sources/chip_seq/remap/gene_tf_pairs_genesymbol.txt'
    ),
    reader = read_tsv,
    reader_param = list(
        col_names = c(
            'source_genesymbol',
            'target_genesymbol',
            'target_ensembl',
            'score'
        ),
        col_types = cols(),
        progress = FALSE
    ),
  resource = 'ReMap'
)
```

Index