Package: tima (via r-universe)

August 23, 2024

Title Taxonomically Informed Metabolite Annotation

Version 2.11.0

Maintainer Adriano Rutz <adafede@gmail.com>

Description This package provides the infrastructure to perform Taxonomically Informed Metabolite Annotation.

License GPL (>= 3)

URL https://github.com/taxonomicallyinformedannotation/tima, https://taxonomicallyinformedannotation.github.io/tima, https://taxonomicallyinformedannotation.github.io/tima-shinylive

BugReports https://github.com/taxonomicallyinformedannotation/tima/issues

Depends R (>= 4.3.0)

Imports crayon (>= 1.5.3), docopt (>= 0.7.1), dplyr (>= 1.1.4), DT, fs (>= 1.6.4), gt, httr (>= 1.4.7), httr2 (>= 1.0.1), igraph (>= 2.0.3), jsonlite (>= 1.8.8), MetaboCoreUtils (>= 1.10.0), MsBackendMgf (>= 1.10.0), MsBackendMsp (>= 1.6.0), msentropy (>= 0.1.4), pbapply (>= 1.7.2), rotl (>= 3.1.0), shiny (>= 1.9.1), shinybusy, shinyWidgets, Spectra (>= 1.12.0), stats, stringi (>= 1.8.4), targets (>= 1.7.1), tidyfst (>= 1.7.8), tidyselect (>= 1.2.1), tidytable (>= 0.11.1), utils, visNetwork, yaml (>= 2.3.10)

Suggests BiocManager, knitr, lifecycle, pkgload, R.utils, rlang, shinyhelper, shinyjs, shinytest2, shinyvalidate, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

biocViews metabolite annotation, chemotaxonomy, scoring system, natural products, computational metabolomics, taxonomic distance, specialized metabolome

ByteCompile true

Config/Needs/website rmarkdown

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

X-schema.org-keywords metabolite annotation, chemotaxonomy, scoring system, natural products, computational metabolomics, taxonomic distance, specialized metabolome

Collate 'round_reals.R' 'log_pipe.R' 'harmonize_adducts.R' 'parse_yaml_params.R' 'parse_cli_params.R' 'get_default_paths.R' 'get_params.R' 'dist_groups.R' 'decorate_masses.R' 'parse_adduct.R' 'calculate_mass_of_m.R' 'annotate_masses.R' 'sanitize_spectra.R' 'import_spectra.R' 'annotate_spectra.R' 'benchmark_taxize_spectra.R' 'calculate_entropy.R' 'clean_bio.R' 'filter_high_confidence_only.R' 'columns_model.R' 'clean_collapse.R' 'clean_chemo.R' 'complement_metadata_structures.R' 'copy_backbone.R' 'create_components.R' 'create_dir.R' 'create_edges.R' 'create_edges_spectra.R' 'decorate_bio.R' 'decorate_chemo.R' 'export_output.R' 'export_params.R' 'export_spectra_rds.R' 'extract_spectra.R' 'fake_annotations_columns.R' 'fake_ecmdb.R' 'fake_hmdb.R' 'fake_lotus.R' 'fake_sop_columns.R' 'filter_annotations.R' 'go_to_cache.R' 'get_file.R' 'get_example_sirius.R' 'get_example_files.R' 'get_gnps_tables.R' 'get_last_version_from_zenodo.R' 'get_massbank_spectra.R' 'get_organism_taxonomy_ott.R' 'globals.R' 'harmonize_names_sirius.R' 'harmonize_spectra.R' 'install.R' 'load_yaml_files.R' 'log_debug.R' 'pre_harmonize_names_sirius.R' 'select_annotations_columns.R' 'prepare_annotations_gnps.R' 'select_sirius_columns.R' 'read_from_sirius_zip.R' 'prepare_annotations_sirius.R' 'prepare_annotations_spectra.R' 'prepare_features_components.R' 'prepare_features_edges.R' 'prepare_features_tables.R' 'prepare_libraries_rt.R' 'select_sop_columns.R' 'prepare_libraries_sop_closed.R' 'prepare_libraries_sop_ecmdb.R' 'prepare_libraries_sop_hmdb.R' 'prepare_libraries_sop_lotus.R' 'split_tables_sop.R' 'prepare_libraries_sop_merged.R' 'prepare_libraries_spectra.R' 'prepare_params.R' 'prepare_taxa.R' 'replace_id.R' 'run_app.R' 'tima-package.R' 'tima_full.R' 'transform_score_sirius_csi.R' 'weight_chemo.R' 'weight_bio.R' 'weight_annotations.R'

Repository https://taxonomicallyinformedannotation.r-universe.dev

RemoteUrl https://github.com/taxonomicallyinformedannotation/tima

RemoteRef HEAD

RemoteSha bfaf53b9f765b8b2ec99b744438272903c84965b

Contents

annotate_masses *Annotate masses*

Description

This function annotates a feature table based on exact mass match. It requires a structural library, its metadata, and lists of adducts, clusters, and neutral losses to be considered. The polarity has to be pos or neg and retention time and mass tolerances should be given. The feature table is expected to be pre-formatted.

Usage

```
annotate_masses(
  features = get_params(step = "annotate_masses")$files$features$prepared,
  output_annotations = get_params(step =
    "annotate_masses")$files$annotations$prepared$structural$ms1,
 output_edges = get_params(step = "annotate_masses")$files$networks$spectral$edges$raw,
  name_source = get_params(step = "annotate_masses")$names$source,
  name_target = get_params(step = "annotate_masses")$names$target,
 library = get_params(step = "annotate_masses")$files$libraries$sop$merged$keys,
  str_stereo = get_params(step =
    "annotate_masses")$files$libraries$sop$merged$structures$stereo,
  str_met = get_params(step =
    "annotate_masses")$files$libraries$sop$merged$structures$metadata,
  str_nam = get_params(step =
    "annotate_masses")$files$libraries$sop$merged$structures$names,
  str_tax_cla = get_params(step =
    "annotate_masses")$files$libraries$sop$merged$structures$taxonomies$cla,
  str_tax_npc = get_params(step =
    "annotate_masses")$files$libraries$sop$merged$structures$taxonomies$npc,
  adducts_list = get_params(step = "annotate_masses")$ms$adducts,
  clusters_list = get_params(step = "annotate_masses")$ms$clusters,
  neutral_losses_list = get_params(step = "annotate_masses")$ms$neutral_losses,
  ms_mode = get_params(step = "annotate_masses")$ms$polarity,
 tolerance_ppm = get_params(step = "annotate_masses")$ms$tolerances$mass$ppm$ms1,
  tolerance_rt = get_params(step = "annotate_masses")$ms$tolerances$rt$adducts
)
```

Arguments

features	Table containing your previous annotation to complement
output_annotations	Output for mass based structural annotations
output_edges	Output for mass based edges
name_source	Name of the source features column
name_target	Name of the target features column

library	Library containing the keys
str_stereo	File containing structures stereo
str_met	File containing structures metadata
str_nam	File containing structures names
str_tax_cla	File containing Classyfire taxonomy
str_tax_npc	File containing NPClassifier taxonomy
adducts_list	List of adducts to be used
clusters_list	List of clusters to be used
neutral_losses_list	List of neutral losses to be used
ms_mode	Ionization mode. Must be 'pos' or 'neg'
tolerance_ppm	Tolerance to perform annotation. Should be <= 20 ppm
tolerance_rt	Tolerance to group adducts. Should be <= 0.05 minutes

Value

The path to the files containing MS1 annotations and edges

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
data_interim <- "data/interim/"
dir <- paste0(github, repo)
dir <- paste0(dir, data_interim)
annotate_masses(
  features = paste0(dir, "features/example_features.tsv"),
  library = paste0(dir, "libraries/sop/merged/keys.tsv"),
  str_stereo = paste0(dir, "libraries/sop/merged/structures/stereo.tsv"),
  str_met = paste0(dir, "libraries/sop/merged/structures/metadata.tsv"),
  str_nam = paste0(dir, "libraries/sop/merged/structures/names.tsv"),
  str_tax_cla = paste0(dir, "libraries/sop/merged/structures/taxonomies/classyfire.tsv"),
  str_tax_npc = paste0(dir, "libraries/sop/merged/structures/taxonomies/npc.tsv")
)
unlink("data", recursive = TRUE)

## End(Not run)
```

annotate_spectra *Annotate spectra*

Description

This function annotates spectra

Usage

```
annotate_spectra(
  input = get_params(step = "annotate_spectra")$files$spectral$raw,
  library = get_params(step = "annotate_spectra")$files$libraries$spectral,
  polarity = get_params(step = "annotate_spectra")$ms$polarity,
 output = get_params(step = "annotate_spectra")$files$annotations$raw$spectral$spectral,
  threshold = get_params(step =
    "annotate_spectra")$annotations$thresholds$ms2$similarity$annotation,
  ppm = get_params(step = "annotate_spectra")$ms$tolerances$mass$ppm$ms2,
  dalton = get_params(step = "annotate_spectra")$ms$tolerances$mass$dalton$ms2,
  qutoff = get_params(step = "annotate_spectra")$ms$thresholds$ms2$intensity,
  approx = get_params(step = "annotate_spectra")$annotations$ms2approx
)
```

Arguments

input	Query file containing spectra. Currently an '.mgf' file
library	Library containing spectra to match against. Can be '.mgf' or '.sqlite' (Spectra formatted)
polarity	MS polarity. Must be 'pos' or 'neg'.
output	Output file.
threshold	Minimal similarity to report
ppm	Relative ppm tolerance to be used
dalton	Absolute Dalton tolerance to be used
qutoff	Intensity under which ms2 fragments will be removed.
approx	Perform matching without precursor match

Details

It takes two files as input. A query file that will be matched against a library file.

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
get_file(
```

```
  url = get_default_paths()$urls$examples$spectra_mini,
  export = get_params(step = "annotate_spectra")$files$spectral$raw
)
get_file(
  url = get_default_paths()$urls$examples$spectral_lib_mini$with_rt,
  export = get_default_paths()$data$source$libraries$spectra$exp$with_rt
)
annotate_spectra(
  library = get_default_paths()$data$source$libraries$spectra$exp$with_rt
)
unlink("data", recursive = TRUE)

## End(Not run)
```

`benchmark_taxize_spectra` *Benchmark taxize spectra*

Description

This function adds taxa to the benchmark

Usage

```
benchmark_taxize_spectra(input, keys, org_tax_ott, output)
```

Arguments

`input`	Initial features
`keys`	SOP keys
`org_tax_ott`	Taxonomy
`output`	Prepared features

Details

Because they are still quite dirty

Value

The path to the taxed benchmark

Examples

```
NULL
```

calculate_entropy *Calculate entropy*

Description

This function calculates entropy similarity between two spectra

Usage

```
calculate_entropy(
  index,
  target,
  frags,
  ms2_tolerance,
  ppm_tolerance,
  threshold = 0.1
)
```

Arguments

index	Index of the first spectrum
target	Index of the second spectrum
frags	List of fragments
ms2_tolerance	MS2 tolerance
ppm_tolerance	ppm tolerance
threshold	Threshold value for the score

Value

A list containing the calculated entropy similarity values or NULL if the score is below the threshold

Examples

```
NULL
```

calculate_mass_of_m *Calculate mass of M*

Description

This function calculates the mass of M

Usage

```
calculate_mass_of_m(adduct_string, mz, electron_mass = 0.0005485799)
```

Arguments

adduct_string	Adduct to be parsed
mz	mz
electron_mass	Electron mass

Value

A mass

Examples

```
calculate_mass_of_m(mz = 123.4567, adduct_string = "[M+H]+")
calculate_mass_of_m(mz = 123.4567, adduct_string = "[M+Na]+")
calculate_mass_of_m(mz = 123.456, adduct_string = "[2M1-C6H12O6 (hexose)+NaCl+H]2+")
```

---

clean_bio *Clean bio*

Description

This function cleans the results obtained after biological weighting

Usage

```
clean_bio(
  annot_table_wei_bio = get("annot_table_wei_bio", envir = parent.frame()),
  edges_table = get("edges_table", envir = parent.frame()),
  minimal_consistency = get("minimal_consistency", envir = parent.frame())
)
```

Arguments

annot_table_wei_bio	Table containing your biologically weighted annotation
edges_table	Table containing the edges between features
minimal_consistency	Minimal consistency score for a class. FLOAT

Value

A table containing the biologically weighted annotation where only a given number of initial candidates are kept

See Also

weight_bio

Examples

```
NULL
```

`clean_chemo` *Clean chemo*

Description

This function cleans the results obtained after chemical weighting

Usage

```
clean_chemo(
  annot_table_wei_chemo = get("annot_table_wei_chemo", envir = parent.frame()),
  components_table = get("components_table", envir = parent.frame()),
  features_table = get("features_table", envir = parent.frame()),
  structure_organism_pairs_table = get("structure_organism_pairs_table", envir =
    parent.frame()),
  candidates_final = get("candidates_final", envir = parent.frame()),
  minimal_ms1_bio = get("minimal_ms1_bio", envir = parent.frame()),
  minimal_ms1_chemo = get("minimal_ms1_chemo", envir = parent.frame()),
  minimal_ms1_condition = get("minimal_ms1_condition", envir = parent.frame()),
  high_confidence = get("high_confidence", envir = parent.frame()),
  remove_ties = get("remove_ties", envir = parent.frame()),
  summarise = get("summarise", envir = parent.frame())
)
```

Arguments

`annot_table_wei_chemo`
: Table containing your chemically weighted annotation

`components_table`
: Prepared components file

`features_table`
: Prepared features file

`structure_organism_pairs_table`
: Table containing the structure - organism pairs

`candidates_final`
: Number of final candidates to keep

`minimal_ms1_bio`
: Minimal biological score to keep MS1 based annotation

`minimal_ms1_chemo`
: Minimal chemical score to keep MS1 based annotation

`minimal_ms1_condition`
: Condition to be used. Must be "OR" or "AND".

`high_confidence`
: Report high confidence candidates only. BOOLEAN

remove_ties	Remove ties. BOOLEAN
summarise	Boolean. summarise results (1 row per feature)

Value

A table containing the chemically weighted annotation where only a given number of initial candidates are kept

See Also

weight_chemo

Examples

```
NULL
```

---

clean_collapse *Clean collapse*

Description

This function collapses a grouped dataframe and trims it

Usage

```
clean_collapse(grouped_df, cols = NA)
```

Arguments

grouped_df	Grouped dataframe
cols	Column(s) to apply collapse to

Value

Cleaned and collapsed dataframe

Examples

```
NULL
```

columns_model *Columns model*

Description

This function models columns

Usage

```
columns_model()
```

Value

The columns model

Examples

```
NULL
```

complement_metadata_structures *Complement metadata of structures*

Description

This function complement structural metadata

Usage

```
complement_metadata_structures(
  df,
  str_stereo = get("str_stereo", envir = parent.frame()),
  str_met = get("str_met", envir = parent.frame()),
  str_nam = get("str_nam", envir = parent.frame()),
  str_tax_cla = get("str_tax_cla", envir = parent.frame()),
  str_tax_npc = get("str_tax_npc", envir = parent.frame())
)
```

Arguments

df	Data frame with structural metadata to be complemented
str_stereo	File containing structures stereo
str_met	File containing structures metadata
str_nam	File containing structures names
str_tax_cla	File containing Classyfire taxonomy
str_tax_npc	File containing NPClassifier taxonomy

Value

Data frame with complemented structural metadata

Examples

```
NULL
```

`copy_backbone` *Copy backbone*

Description

This function copies backbone

Usage

```
copy_backbone(cache_dir = fs::path_home(".tima"), package = "tima")
```

Arguments

`cache_dir`	Cache directory
`package`	Package

Examples

```
NULL
```

`create_components` *Create components*

Description

This function create components from edges

Usage

```
create_components(
  input = get_params(step = "create_components")$files$networks$spectral$edges$prepared,
  output = get_params(step = "create_components")$files$networks$spectral$components$raw
)
```

Arguments

`input`	Input file(s) containing edges
`output`	Output file.

Value

The path to the created components

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
data_interim <- "data/interim/"
dir <- paste0(github, repo)
dir <- paste0(dir, data_interim)
get_file(
  url = paste0(dir, "features/example_edges.tsv"),
  export = get_params(step = "create_components")$files$networks$spectral$edges$prepared
)
create_components()
unlink("data", recursive = TRUE)

## End(Not run)
```

---

create_dir *Create directory*

Description

This function creates a directory at the specified path if it does not already exist.

Usage

```
create_dir(export)
```

Arguments

export	Path to the directory to be created

Value

Message indicating the status of directory creation

Examples

```
create_dir(export = "path/to/directory_of_file")
unlink("path", recursive = TRUE)
```

create_edges *Create edges*

Description

This function applies similarity calculation to a list of spectra to create edges

Usage

```
create_edges(
  index,
  frags,
  precs,
  nspecs,
  ms2_tolerance,
  ppm_tolerance,
  threshold
)
```

Arguments

index	Indices
frags	Fragments
precs	Precursors
nspecs	Number of spectra
ms2_tolerance	MS2 tolerance
ppm_tolerance	ppm tolerance
threshold	Threshold

Examples

```
NULL
```

create_edges_spectra *Create edges spectra*

Description

This function create edges based on fragmentation spectra similarity

Usage

```
create_edges_spectra(
  input = get_params(step = "create_edges_spectra")$files$spectral$raw,
 output = get_params(step = "create_edges_spectra")$files$networks$spectral$edges$raw,
  name_source = get_params(step = "create_edges_spectra")$names$source,
  name_target = get_params(step = "create_edges_spectra")$names$target,
  threshold = get_params(step =
    "create_edges_spectra")$annotations$thresholds$ms2$similarity$edges,
  ppm = get_params(step = "create_edges_spectra")$ms$tolerances$mass$ppm$ms2,
 dalton = get_params(step = "create_edges_spectra")$ms$tolerances$mass$dalton$ms2,
  qutoff = get_params(step = "create_edges_spectra")$ms$thresholds$ms2$intensity
)
```

Arguments

input	Query file containing spectra. Currently an '.mgf' file
output	Output file.
name_source	Name of the source features column
name_target	Name of the target features column
threshold	Minimal similarity to report
ppm	Relative ppm tolerance to be used
dalton	Absolute Dalton tolerance to be used
qutoff	Intensity under which ms2 fragments will be removed.

Value

The path to the created spectral edges

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
get_file(
  url = get_default_paths()$urls$examples$spectra_mini,
  export = get_params(step = "create_edges_spectra")$files$spectral$raw
)
create_edges_spectra()
unlink("data", recursive = TRUE)

## End(Not run)
```

decorate_bio *Decorate bio*

Description

This function outputs information about biological weighting

Usage

```
decorate_bio(
  annot_table_wei_bio = get("annot_table_wei_chemo", envir = parent.frame()),
 score_biological_kingdom = get("score_biological_kingdom", envir = parent.frame()),
 score_biological_phylum = get("score_biological_phylum", envir = parent.frame()),
 score_biological_class = get("score_biological_class", envir = parent.frame()),
 score_biological_order = get("score_biological_order", envir = parent.frame()),
 score_biological_family = get("score_biological_family", envir = parent.frame()),
 score_biological_tribe = get("score_biological_tribe", envir = parent.frame()),
 score_biological_genus = get("score_biological_genus", envir = parent.frame()),
 score_biological_species = get("score_biological_species", envir = parent.frame()),
 score_biological_variety = get("score_biological_variety", envir = parent.frame())
)
```

Arguments

annot_table_wei_bio	Table to decorate
score_biological_kingdom	Kingdom score
score_biological_phylum	Phylum score
score_biological_class	Class score
score_biological_order	Order score
score_biological_family	Family score
score_biological_tribe	Tribe score
score_biological_genus	Genus score
score_biological_species	Species score
score_biological_variety	Variety score

Value

Message indicating the number of annotations weighted at each biological level

Examples

```
NULL
```

`decorate_chemo` *Decorate chemo*

Description

This function outputs information about chemical weighting

Usage

```
decorate_chemo(
  annot_table_wei_chemo = get("annot_table_wei_chemo", envir = parent.frame()),
  score_chemical_cla_kingdom = get("score_chemical_cla_kingdom", envir = parent.frame()),
  score_chemical_cla_superclass = get("score_chemical_cla_superclass", envir =
    parent.frame()),
  score_chemical_cla_class = get("score_chemical_cla_class", envir = parent.frame()),
  score_chemical_cla_parent = get("score_chemical_cla_parent", envir = parent.frame()),
  score_chemical_npc_pathway = get("score_chemical_npc_pathway", envir = parent.frame()),
  score_chemical_npc_superclass = get("score_chemical_npc_superclass", envir =
    parent.frame()),
  score_chemical_npc_class = get("score_chemical_npc_class", envir = parent.frame())
)
```

Arguments

`annot_table_wei_chemo`
: Table to decorate

`score_chemical_cla_kingdom`
: Classyfire kingdom score

`score_chemical_cla_superclass`
: Classyfire superclass score

`score_chemical_cla_class`
: Classyfire class score

`score_chemical_cla_parent`
: Classyfire parent score

`score_chemical_npc_pathway`
: NPC pathway score

`score_chemical_npc_superclass`
: NPC superclass score

`score_chemical_npc_class`
: NPC class score

Value

Message indicating the number of annotations weighted at each chemical level

Examples

```
NULL
```

`decorate_masses` *Decorate masses*

Description

This function outputs information about MS1 annotation

Usage

```
decorate_masses(
  annotation_table_ms1 = get("annotation_table_ms1", envir = parent.frame())
)
```

Arguments

`annotation_table_ms1`
: Table to decorate

Value

Message indicating the number of annotations obtained by MS1

Examples

```
NULL
```

`dist_get` *Distance between two elements in a distance matrix*

Description

This function calculates the distance between two elements in a distance matrix

Usage

```
dist_get(d, idx1, idx2)
```

Arguments

d	Distance matrix
idx1	Index of the first element
idx2	Index of the second element

Details

Credit goes to usedist package

Value

Distance between the two elements

Examples

```
NULL
```

---

dist_groups *Dist groups*

Description

This function gets distances per group

Usage

```
dist_groups(d, g)
```

Arguments

d	A distance object
g	A grouping vector for the distance object

Value

A data frame containing distance information between pairs of observations in the distance object, with columns for the names or indices of the observations, the group labels for each observation, and the distance between the observations. The label column indicates whether the distance is within a group or between groups.

Examples

```
NULL
```

export_output *Export output*

Description

This function creates the output directory if it doesn't exist and exports the data frame to a tab-delimited file.

Usage

```
export_output(x, file = output)
```

Arguments

x	data frame to be exported
file	path to the output file

Value

The path of the exported file

Examples

```
export_output(x = data.frame(), file = "output/file.tsv")
unlink("output", recursive = TRUE)
```

export_params *Export parameters*

Description

This function writes the parameters to a YAML file in the specified directory.

Usage

```
export_params(
  parameters = get("parameters", envir = parent.frame()),
  directory = get_default_paths()$data$interim$params$path,
  step
)
```

Arguments

parameters	list of parameters to be exported
directory	directory where the YAML file will be saved
step	step identifier to be included in the YAML file name

Examples

```
NULL
```

`export_spectra_rds` *Export spectra RDS*

Description

This function exports spectra in RDS

Usage

```
export_spectra_rds(file, spectra)
```

Arguments

`file`	File where spectra will be exported.
`spectra`	The spectra object where spectra are stored

Examples

```
NULL
```

`extract_spectra` *Extract spectra from a Spectra object*

Description

This function extracts spectra from a `Spectra`object

Usage

```
extract_spectra(object)
```

Arguments

`object`	Object of class Spectra

Value

Data frame containing spectra data

Examples

```
NULL
```

fake_annotations_columns *Fake annotations columns*

Description

This function fakes annotations columns

Usage

```
fake_annotations_columns()
```

Examples

```
NULL
```

fake_ecmdb *Fake ECMDB*

Description

This function fakes ECMDB in case the download failed

Usage

```
fake_ecmdb(export)
```

Arguments

export	Path to save the file to

Examples

```
NULL
```

fake_hmdb *Fake HMDB*

Description

This function fakes HMDB in case the download failed

Usage

```
fake_hmdb(export)
```

Arguments

export	Path to save the file to

Examples

```
NULL
```

fake_lotus *Fake LOTUS*

Description

This function fakes LOTUS in case the download failed

Usage

```
fake_lotus(export)
```

Arguments

export	Path to save the file to

Examples

```
NULL
```

fake_sop_columns *Fake SOP columns*

Description

This function fakes sop columns

Usage

```
fake_sop_columns()
```

Examples

```
NULL
```

filter_annotations *Filter annotations*

Description

This function filters initial annotations.

Usage

```
filter_annotations(
  annotations = get_params(step =
    "filter_annotations")$files$annotations$prepared$structural,
  features = get_params(step = "filter_annotations")$files$features$prepared,
 rts = get_params(step = "filter_annotations")$files$libraries$temporal$prepared,
  output = get_params(step = "filter_annotations")$files$annotations$filtered,
 tolerance_rt = get_params(step = "filter_annotations")$ms$tolerances$rt$library
)
```

Arguments

annotations	Prepared annotations file
features	Prepared features file
rts	Prepared retention time library
output	Output file
tolerance_rt	Tolerance to filter retention time

Value

The path to the filtered annotations

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
dir <- paste0(github, repo)
annotations <- get_params(step = "filter_annotations")$files$annotations$prepared$structural[[2]] |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
features <- get_params(step = "filter_annotations")$files$features$prepared |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
rts <- get_params(step = "filter_annotations")$files$libraries$temporal$prepared |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
get_file(url = paste0(dir, annotations), export = annotations)
get_file(url = paste0(dir, features), export = features)
get_file(url = paste0(dir, rts), export = rts)
filter_annotations(
  annotations = annotations,
  features = features,
  rts = rts
)
unlink("data", recursive = TRUE)

## End(Not run)
```

`filter_high_confidence_only`

Filter high confidence only **[Experimental]**

Description

This function filters highly confident annotations only.

Usage

```
filter_high_confidence_only(df, score_bio_min = 0.85, score_ini_min = 0.75)
```

Arguments

df	Dataframe
score_bio_min	Minimal biological score. Current default to 0.85.
score_ini_min	Minimal initial score. Current default to 0.75.

Value

A dataframe containing only high confidence annotations

Examples

```
NULL
```

---

get_default_paths *Get default paths*

Description

This function gets default paths

Usage

```
get_default_paths(yaml = system.file("paths.yaml", package = "tima"))
```

Arguments

yaml	The YAML file containing the paths (default is "paths.yaml")

Value

A list containing the paths specified in the YAML file

Examples

```
get_default_paths()
```

get_example_files *Get example files*

Description

This function downloads example files

Usage

```
get_example_files(
  example = c("features", "metadata", "sirius", "spectra"),
  in_cache = TRUE
)
```

Arguments

example	The example(s) you want to download
in_cache	Flag to indicate if storing the files in cache

Value

Example files.

Examples

```
get_example_files(example = c("features"), in_cache = FALSE)
unlink("data", recursive = TRUE)
```

get_example_sirius *Get example sirius*

Description

This function gets example SIRIUS annotations

Usage

```
get_example_sirius(
  url = get_default_paths()$urls$examples$sirius,
  export = get_default_paths()$data$interim$annotations$example_sirius
)
```

Arguments

url	URL where the example is accessible
export	Path where to save the example

Examples

```
NULL
```

`get_file` *Get file*

Description

This function get files

Usage

```
get_file(url, export, limit = 3600)
```

Arguments

`url`	URL of the file to be downloaded
`export`	File path where the file should be saved
`limit`	Timeout limit (in seconds)

Value

The path to the file

Examples

```
git <- "https://github.com/"
org <- "taxonomicallyinformedannotation"
repo <- "tima-example-files"
branch <- "main"
file <- "example_metadata.tsv"
get_file(
  url = paste(git, org, repo, "raw", branch, file, sep = "/"),
  export = "data/source/example_metadata.tsv"
)
unlink("data", recursive = TRUE)
```

get_gnps_tables *Get GNPS Tables*

Description

This function gets GNPS tables from corresponding job ID.

Usage

```
get_gnps_tables(
  gnps_job_id,
  gnps_job_example = get_default_paths()$gnps$example,
  filename,
  workflow = "fbmn",
  path_features,
  path_metadata,
  path_spectra,
  path_source = get_default_paths()$data$source$path,
  path_interim_a = get_default_paths()$data$interim$annotations$path,
  path_interim_f = get_default_paths()$data$interim$features$path
)
```

Arguments

gnps_job_id	GNPS job ID
gnps_job_example	GNPS job example
filename	Name of the file
workflow	Character string indicating the type of workflow, either "fbmn" or "classical"
path_features	Path to features
path_metadata	Path to metadata
path_spectra	Path to spectra
path_source	Path to store the source files
path_interim_a	Path to store the interim annotations file
path_interim_f	Path to store the interim features files

Value

The downloaded GNPS tables

Examples

```
NULL
```

get_last_version_from_zenodo *Get last version from Zenodo*

Description

This function gets the last version of a file from a Zenodo record

Usage

```
get_last_version_from_zenodo(doi, pattern, path)
```

Arguments

doi	DOI of the Zenodo record
pattern	Pattern to identify the file to download
path	Path to save the file to

Details

Credit goes to partially to https://inbo.github.io/inborutils/

Value

The path to the file

Examples

```
get_last_version_from_zenodo(
  doi = "10.5281/zenodo.5794106",
  pattern = "frozen.csv.gz",
  path = "frozen.csv.gz"
)
unlink("frozen.csv.gz")
```

get_massbank_spectra *Get MassBank spectra*

Description

This function gets MassBank spectra

Usage

```
get_massbank_spectra(
  output_dir = "data/source/libraries/spectra/exp",
  mb_file = get_default_paths()$urls$massbank$file,
  mb_url = get_default_paths()$urls$massbank$url,
  mb_version = get_default_paths()$urls$massbank$version
)
```

Arguments

output_dir	Output where to store the spectra
mb_file	MassBank file
mb_url	MassBank URL
mb_version	MassBank version

Value

The path to MassBank spectra

Examples

```
NULL
```

---

get_organism_taxonomy_ott

Get organism taxonomy (Open Tree of Life Taxonomy)

Description

This function retrieves taxonomy from the Open Tree of Life taxonomy

Usage

```
get_organism_taxonomy_ott(
  df,
  url = "https://api.opentreeoflife.org/v3/taxonomy/about",
  retry = TRUE
)
```

Arguments

df	Dataframe containing your organism(s) name(s)
url	url of the ott api (for testing purposes)
retry	Boolean. Retry with generic epithet

Value

The path to the obtained OTT taxonomy

Examples

```
df <- data.frame("organism" = "Homo sapiens")
get_organism_taxonomy_ott(df)
unlink("data", recursive = TRUE)
```

`get_params` *Get parameters*

Description

This function gets the parameters for the job. Combination of cli and yaml parameters

Usage

```
get_params(step)
```

Arguments

`step`	Name of the step being performed

Value

The parameters

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
get_params("prepare_params")

## End(Not run)
```

go_to_cache *Go to cache*

Description

This function goes to cache

Usage

```
go_to_cache(dir = ".tima")
```

Arguments

dir	Directory

Value

Goes to cache

Examples

```
NULL
```

harmonize_adducts *Harmonize adducts*

Description

This function annotates masses

Usage

```
harmonize_adducts(df, adducts_colname = "adduct")
```

Arguments

df	Dataframe
adducts_colname	Adducts colname

Value

A table with harmonized adducts

Examples

```
NULL
```

harmonize_names_sirius *Harmonize names sirius*

Description

This function harmonizes the names of Sirius outputs to make them compatible

Usage

```
harmonize_names_sirius(x)
```

Arguments

x	Character string containing a name

Value

Character string with the name modified according to the rules specified in the function

Examples

```
harmonized_name <- tima:::harmonize_names_sirius("My_name")
```

harmonize_spectra *Harmonize spectra*

Description

This function harmonizes spectra headers

Usage

```
harmonize_spectra(
  spectra,
  metad = get("metad", envir = parent.frame()),
  mode,
  col_ad = get("col_ad", envir = parent.frame()),
  col_ce = get("col_ce", envir = parent.frame()),
  col_ci = get("col_ci", envir = parent.frame()),
  col_em = get("col_em", envir = parent.frame()),
  col_in = get("col_in", envir = parent.frame()),
  col_io = get("col_io", envir = parent.frame()),
  col_ik = get("col_ik", envir = parent.frame()),
  col_il = get("col_il", envir = parent.frame()),
  col_mf = get("col_mf", envir = parent.frame()),
```

```
  col_na = get("col_na", envir = parent.frame()),
  col_po = get("col_po", envir = parent.frame()),
  col_sm = get("col_sm", envir = parent.frame()),
  col_sn = get("col_sn", envir = parent.frame()),
  col_si = get("col_si", envir = parent.frame()),
  col_sp = get("col_sp", envir = parent.frame()),
  col_sy = get("col_sy", envir = parent.frame()),
  col_xl = get("col_xl", envir = parent.frame())
)
```

Arguments

spectra	Spectra object to be harmonized
metad	Metadata to identify the library
mode	MS ionization mode. Must contain 'pos' or 'neg'
col_ad	Name of the adduct in mgf
col_ce	Name of the collision energy in mgf
col_ci	Name of the compound id in mgf
col_em	Name of the exact mass in mgf
col_in	Name of the InChI in mgf
col_io	Name of the InChI without stereo in mgf
col_ik	Name of the InChIKey in mgf
col_il	Name of the InChIKey without stereo in mgf
col_mf	Name of the molecular formula in mgf
col_na	Name of the name in mgf
col_po	Name of the polarity in mgf
col_sm	Name of the SMILES in mgf
col_sn	Name of the SMILES without stereo in mgf
col_si	Name of the spectrum id in mgf
col_sp	Name of the SPLASH in mgf
col_sy	Name of the synonyms in mgf
col_xl	Name of the xlogp in mgf

Value

The harmonized spectra

Examples

```
NULL
```

`import_spectra` *Import spectra*

Description

This function imports spectra from a file (.mgf or .sqlite)

Usage

```
import_spectra(
  file,
  cutoff = 0,
  dalton = 0.01,
  polarity = NA,
  ppm = 10,
  sanitize = TRUE
)
```

Arguments

`file`	File path of the spectrum file to be imported
`cutoff`	Absolute minimal intensity
`dalton`	Dalton tolerance
`polarity`	Polarity
`ppm`	PPM tolerance
`sanitize`	Flag indicating whether to sanitize. Default TRUE

Value

Spectra object containing the imported spectra

Examples

```
get_file(
  url = get_default_paths()$urls$examples$spectra_mini,
  export = get_default_paths()$data$source$spectra
)
import_spectra(file = get_default_paths()$data$source$spectra)
import_spectra(
  file = get_default_paths()$data$source$spectra,
  sanitize = FALSE
)
```

install *Install*

Description

This function runs some required install

Usage

```
install(
  package = "tima",
  repos = c("https://taxonomicallyinformedannotation.r-universe.dev",
    "https://bioc.r-universe.dev", "https://cloud.r-project.org"),
  dependencies = TRUE,
  test = FALSE
)
```

Arguments

package	Package
repos	Repos
dependencies	Flag for dependencies
test	Flag for tests

Examples

```
NULL
```

load_yaml_files *Load yaml files*

Description

This function load yaml files

Usage

```
load_yaml_files()
```

Value

A list of loaded yaml files

Examples

```
NULL
```

log_debug *Log debug*

Description

Simple helper for debugging

Usage

```
log_debug(...)
```

Arguments

...	one or more values to be logged

Value

Message for debugging

Examples

```
log_debug("This is a debug message")
```

log_pipe *Log pipe*

Description

Simple helper for debugging between pipes

Usage

```
log_pipe(x, ...)
```

Arguments

x	value for the pipe
...	one or more values to be logged

Value

Message for debugging

Examples

```
NULL
```

parse_adduct *Parse adduct*

Description

This function parses adducts

Usage

```
parse_adduct(
  adduct_string,
  regex = "\\[(\\d*)M(?![a-z])(\\d*)([+-][\\w\\d].*)?.*\\](\\d*)([+-])?"
)
```

Arguments

adduct_string	Adduct to be parsed
regex	Regex used for parsing

Value

Parsed elements from adduct

Examples

```
parse_adduct("[M+H]+")
parse_adduct("[2M1-C6H12O6 (hexose)+NaCl+H]2+")
```

parse_cli_params *Parse CLI parameters*

Description

This function parses command line parameters

Usage

```
parse_cli_params(arguments, parameters)
```

Arguments

arguments	CLI arguments
parameters	Parameters

Value

Parameters coming from the CLI

Examples

```
NULL
```

`parse_yaml_params` *Parse YAML parameters*

Description

This function parses YAML parameters

Usage

```
parse_yaml_params(
  def = get("default_path", envir = parent.frame()),
  usr = get("user_path", envir = parent.frame())
)
```

Arguments

`def`	Default path
`usr`	User path

Value

A list containing the parameters specified in the YAML files

Examples

```
NULL
```

prepare_annotations_gnps *Prepare annotations GNPS*

Description

This function prepares GNPS obtained annotations

Usage

```
prepare_annotations_gnps(
  input = get_params(step =
    "prepare_annotations_gnps")$files$annotations$raw$spectral$gnps,
  output = get_params(step =
    "prepare_annotations_gnps")$files$annotations$prepared$structural$gnps,
  str_stereo = get_params(step =
    "prepare_annotations_gnps")$files$libraries$sop$merged$structures$stereo,
  str_met = get_params(step =
    "prepare_annotations_gnps")$files$libraries$sop$merged$structures$metadata,
  str_nam = get_params(step =
    "prepare_annotations_gnps")$files$libraries$sop$merged$structures$names,
  str_tax_cla = get_params(step =
   "prepare_annotations_gnps")$files$libraries$sop$merged$structures$taxonomies$cla,
  str_tax_npc = get_params(step =
   "prepare_annotations_gnps")$files$libraries$sop$merged$structures$taxonomies$npc
)
```

Arguments

input	Input file
output	Output file
str_stereo	File containing structures stereo
str_met	File containing structures metadata
str_nam	File containing structures names
str_tax_cla	File containing Classyfire taxonomy
str_tax_npc	File containing NPClassifier taxonomy

Value

The path to the prepared GNPS annotations

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
prepare_annotations_gnps()
unlink("data", recursive = TRUE)

## End(Not run)
```

`prepare_annotations_sirius` *Prepare annotations SIRIUS*

Description

This function prepares Sirius results to make them compatible

Usage

```
prepare_annotations_sirius(
  input_directory = get_params(step =
    "prepare_annotations_sirius")$files$annotations$raw$sirius,
  output_ann = get_params(step =
    "prepare_annotations_sirius")$files$annotations$prepared$structural$sirius,
  output_can = get_params(step =
    "prepare_annotations_sirius")$files$annotations$prepared$canopus,
  output_for = get_params(step =
    "prepare_annotations_sirius")$files$annotations$prepared$formula,
 sirius_version = get_params(step = "prepare_annotations_sirius")$tools$sirius$version,
  str_stereo = get_params(step =
    "prepare_annotations_sirius")$files$libraries$sop$merged$structures$stereo,
  str_met = get_params(step =
   "prepare_annotations_sirius")$files$libraries$sop$merged$structures$metadata,
  str_nam = get_params(step =
    "prepare_annotations_sirius")$files$libraries$sop$merged$structures$names,
  str_tax_cla = get_params(step =
   "prepare_annotations_sirius")$files$libraries$sop$merged$structures$taxonomies$cla,
  str_tax_npc = get_params(step =
   "prepare_annotations_sirius")$files$libraries$sop$merged$structures$taxonomies$npc
)
```

Arguments

input_directory	Directory containing the Sirius results
output_ann	Output where to save prepared annotation results
output_can	Output where to save prepared canopus results

output_for	Output where to save prepared formula results
sirius_version	Sirius version
str_stereo	File containing structures stereo
str_met	File containing structures metadata
str_nam	File containing structures names
str_tax_cla	File containing Classyfire taxonomy
str_tax_npc	File containing NPClassifier taxonomy

Value

The path to the prepared SIRIUS annotations

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
prepare_annotations_sirius()
unlink("data", recursive = TRUE)

## End(Not run)
```

---

prepare_annotations_spectra

Prepare annotations MS2

Description

This function prepares the spectral matches obtained previously to make them compatible

Usage

```
prepare_annotations_spectra(
  input = get_params(step =
    "prepare_annotations_spectra")$files$annotations$raw$spectral$spectral,
  output = get_params(step =
   "prepare_annotations_spectra")$files$annotations$prepared$structural$spectral,
  str_stereo = get_params(step =
   "prepare_annotations_spectra")$files$libraries$sop$merged$structures$stereo,
  str_met = get_params(step =
   "prepare_annotations_spectra")$files$libraries$sop$merged$structures$metadata,
  str_nam = get_params(step =
    "prepare_annotations_spectra")$files$libraries$sop$merged$structures$names,
  str_tax_cla = get_params(step =
   "prepare_annotations_spectra")$files$libraries$sop$merged$structures$taxonomies$cla,
  str_tax_npc = get_params(step =
   "prepare_annotations_spectra")$files$libraries$sop$merged$structures$taxonomies$npc
)
```

Arguments

input	Input file
output	Output file
str_stereo	File containing structures stereo
str_met	File containing structures metadata
str_nam	File containing structures names
str_tax_cla	File containing Classyfire taxonomy
str_tax_npc	File containing NPClassifier taxonomy

Value

The path to the prepared spectral annotations

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
data_interim <- "data/interim/"
dir <- paste0(github, repo)
input <- get_params(step = "prepare_annotations_spectra")$files$annotations$raw$spectral$spectral |>
  gsub(
    pattern = ".tsv.gz",
    replacement = "_pos.tsv",
    fixed = TRUE
  )
get_file(url = paste0(dir, input), export = input)
dir <- paste0(dir, data_interim)
prepare_annotations_spectra(
  input = input,
  str_stereo = paste0(dir, "libraries/sop/merged/structures/stereo.tsv"),
  str_met = paste0(dir, "libraries/sop/merged/structures/metadata.tsv"),
  str_nam = paste0(dir, "libraries/sop/merged/structures/names.tsv"),
 str_tax_cla = paste0(dir, "libraries/sop/merged/structures/taxonomies/classyfire.tsv"),
  str_tax_npc = paste0(dir, "libraries/sop/merged/structures/taxonomies/npc.tsv")
)
unlink("data", recursive = TRUE)

## End(Not run)
```

prepare_features_components *Prepare features components*

Description

This function prepares the components (clusters in molecular network) for further use

Usage

```
prepare_features_components(
  input = get_params(step =
    "prepare_features_components")$files$networks$spectral$components$raw,
  output = get_params(step =
    "prepare_features_components")$files$networks$spectral$components$prepared
)
```

Arguments

input	Input file
output	Output file

Value

The path to the prepared features' components

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
dir <- paste0(github, repo)
input <- get_params(step = "prepare_features_components")$files$networks$spectral$components$raw
get_file(url = paste0(dir, input), export = input)
prepare_features_components(
  input = input
)
unlink("data", recursive = TRUE)

## End(Not run)
```

prepare_features_edges *Prepare features edges*

Description

This function prepares edges for further use

Usage

```
prepare_features_edges(
  input = get_params(step = "prepare_features_edges")$files$networks$spectral$edges$raw,
  output = get_params(step =
    "prepare_features_edges")$files$networks$spectral$edges$prepared,
  name_source = get_params(step = "prepare_features_edges")$names$source,
  name_target = get_params(step = "prepare_features_edges")$names$target
)
```

Arguments

input	Input file if 'manual'
output	Output file
name_source	Name of the source features column
name_target	Name of the target features column

Value

The path to the prepared edges

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
dir <- paste0(github, repo)
input_1 <- get_params(step = "prepare_features_edges")$files$networks$spectral$edges$raw$ms1
input_2 <- get_params(step = "prepare_features_edges")$files$networks$spectral$edges$raw$spectral
get_file(url = paste0(dir, input_1), export = input_1)
get_file(url = paste0(dir, input_2), export = input_2)
prepare_features_edges(
  input = list("ms1" = input_1, "spectral" = input_2)
)
unlink("data", recursive = TRUE)

## End(Not run)
```

prepare_features_tables *Prepare features table*

Description

This function prepares features

Usage

```
prepare_features_tables(
  features = get_params(step = "prepare_features_tables")$files$features$raw,
  output = get_params(step = "prepare_features_tables")$files$features$prepared,
  name_adduct = get_params(step = "prepare_features_tables")$names$adduct,
  name_features = get_params(step = "prepare_features_tables")$names$features,
  name_rt = get_params(step = "prepare_features_tables")$names$rt$features,
  name_mz = get_params(step = "prepare_features_tables")$names$precursor
)
```

Arguments

features	Path to the file containing the features data
output	Path to the file to export the merged data to
name_adduct	Name of the adduct column in the features data
name_features	Name of the features column in the features data
name_rt	Name of the retention time column in the features data
name_mz	Name of the m/z column in the features data

Value

The path to the prepared feature table

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
get_file(
  url = get_default_paths()$urls$examples$features,
  export = get_params(step = "prepare_features_tables")$files$features$raw
)
prepare_features_tables()
unlink("data", recursive = TRUE)

## End(Not run)
```

`prepare_libraries_rt` *Prepare libraries of retention times*

Description

This function prepares retention times libraries to be used for later

Usage

```
prepare_libraries_rt(
 mgf_exp = get_params(step = "prepare_libraries_rt")$files$libraries$temporal$exp$mgf,
 mgf_is = get_params(step = "prepare_libraries_rt")$files$libraries$temporal$is$mgf,
 temp_exp = get_params(step = "prepare_libraries_rt")$files$libraries$temporal$exp$csv,
 temp_is = get_params(step = "prepare_libraries_rt")$files$libraries$temporal$is$csv,
 output_rt = get_params(step = "prepare_libraries_rt")$files$libraries$temporal$prepared,
 output_sop = get_params(step = "prepare_libraries_rt")$files$libraries$sop$prepared$rt,
  col_ik = get_params(step = "prepare_libraries_rt")$names$mgf$inchikey,
  col_rt = get_params(step = "prepare_libraries_rt")$names$mgf$retention_time,
  col_sm = get_params(step = "prepare_libraries_rt")$names$mgf$smiles,
  name_inchikey = get_params(step = "prepare_libraries_rt")$names$inchikey,
  name_rt = get_params(step = "prepare_libraries_rt")$names$rt$library,
  name_smiles = get_params(step = "prepare_libraries_rt")$names$smiles,
  unit_rt = get_params(step = "prepare_libraries_rt")$units$rt
)
```

Arguments

`mgf_exp`	MGF containing experimental retention times
`mgf_is`	MGF containing in silico predicted retention times
`temp_exp`	File containing experimental retention times
`temp_is`	File containing in silico predicted retention times
`output_rt`	Output retention time file
`output_sop`	Output pseudo sop file
`col_ik`	Name of the InChIKey in mgf
`col_rt`	Name of the retention time in mgf
`col_sm`	Name of the SMILES in mgf
`name_inchikey`	Name of the InChIKey in file
`name_rt`	Name of the retention time in file
`name_smiles`	Name of the SMILES in file
`unit_rt`	Unit of the retention time. Must be "seconds" or "minutes"

Value

The path to the prepared retention time library

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
prepare_libraries_rt()
unlink("data", recursive = TRUE)

## End(Not run)
```

`prepare_libraries_sop_closed` *Prepare libraries of structure organism pairs CLOSED*

Description

Prepare libraries of structure organism pairs CLOSED

Usage

```
prepare_libraries_sop_closed(
  input = get_params(step =
    "prepare_libraries_sop_closed")$files$libraries$sop$raw$closed,
  output = get_params(step =
    "prepare_libraries_sop_closed")$files$libraries$sop$prepared$closed
)
```

Arguments

`input`	Input file
`output`	Output file

Value

The path to the prepared structure-organism pairs library CLOSED

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
prepare_libraries_sop_closed()
unlink("data", recursive = TRUE)

## End(Not run)
```

prepare_libraries_sop_ecmdb

Prepare libraries of structure organism pairs ECMDB

Description

Prepare libraries of structure organism pairs ECMDB

Usage

```
prepare_libraries_sop_ecmdb(
  input = get_params(step = "prepare_libraries_sop_ecmdb")$files$libraries$sop$raw$ecmdb,
  output = get_params(step =
    "prepare_libraries_sop_ecmdb")$files$libraries$sop$prepared$ecmdb
)
```

Arguments

input	Input file
output	Output file

Value

The path to the prepared structure-organism pairs library ECMDB

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
prepare_libraries_sop_ecmdb()
unlink("data", recursive = TRUE)

## End(Not run)
```

prepare_libraries_sop_hmdb

Prepare libraries of structure organism pairs HMDB

Description

This function prepares the HMDB structure-organism pairs

Usage

```
prepare_libraries_sop_hmdb(
  input = get_params(step = "prepare_libraries_sop_hmdb")$files$libraries$sop$raw$hmdb,
  output = get_params(step =
    "prepare_libraries_sop_hmdb")$files$libraries$sop$prepared$hmdb
)
```

Arguments

input	Input file
output	Output file

Value

The path to the prepared structure-organism pairs library HMDB

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
prepare_libraries_sop_hmdb()
unlink("data", recursive = TRUE)

## End(Not run)
```

---

prepare_libraries_sop_lotus

Prepare libraries of structure organism pairs LOTUS

Description

This function prepares the LOTUS structure-organism pairs

Usage

```
prepare_libraries_sop_lotus(
  input = get_params(step = "prepare_libraries_sop_lotus")$files$libraries$sop$raw$lotus,
  output = get_params(step =
    "prepare_libraries_sop_lotus")$files$libraries$sop$prepared$lotus
)
```

Arguments

input	Input file
output	Output file

Value

The path to the prepared structure-organism pairs library LOTUS

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
prepare_libraries_sop_lotus()
unlink("data", recursive = TRUE)

## End(Not run)
```

---

prepare_libraries_sop_merged

Prepare merged structure organism pairs libraries

Description

This function prepares the libraries made of all sub-libraries containing structure-organism pairs

Usage

```
prepare_libraries_sop_merged(
  files = get_params(step = "prepare_libraries_sop_merged")$files$libraries$sop$prepared,
  filter = get_params(step = "prepare_libraries_sop_merged")$organisms$filter$mode,
  level = get_params(step = "prepare_libraries_sop_merged")$organisms$filter$level,
  value = get_params(step = "prepare_libraries_sop_merged")$organisms$filter$value,
   output_key = get_params(step =
     "prepare_libraries_sop_merged")$files$libraries$sop$merged$keys,
   output_org_tax_ott = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$organisms$taxonomies$ott,
   output_str_stereo = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$structures$stereo,
   output_str_met = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$structures$metadata,
   output_str_nam = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$structures$names,
   output_str_tax_cla = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$structures$taxonomies$cla,
   output_str_tax_npc = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$structures$taxonomies$npc
)
```

Arguments

files	List of libraries to be merged
filter	Boolean. TRUE or FALSE if you want to filter the library
level	Biological rank to be filtered. Kingdom, phylum, family, genus, ...
value	Name of the taxon or taxa to be kept, e.g. 'GentianaceaelApocynaceae'
output_key	Output file for keys
output_org_tax_ott	Output file for organisms taxonomy (OTT)
output_str_stereo	Output file for structures stereo
output_str_met	Output file for structures metadata
output_str_nam	Output file for structures names
output_str_tax_cla	Output file for structures taxonomy (Classyfire)
output_str_tax_npc	Output file for structures taxonomy (NPC)

Details

It can be restricted to specific taxa to have more biologically meaningful annotation.

Value

The path to the prepared structure-organism pairs library MERGED

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
dir <- paste0(github, repo)
files <- get_params(step = "prepare_libraries_sop_merged")$files$libraries$sop$prepared$lotus |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
get_file(url = paste0(dir, files), export = files)
prepare_libraries_sop_merged(files = files)
unlink("data", recursive = TRUE)

## End(Not run)
```

`prepare_libraries_spectra` *Prepare libraries of spectra*

Description

This function prepares spectra to be used for spectral matching

Usage

```
prepare_libraries_spectra(
  input = get_params(step = "prepare_libraries_spectra")$files$libraries$spectral$raw,
   nam_lib = get_params(step = "prepare_libraries_spectra")$names$libraries,
   col_ad = get_params(step = "prepare_libraries_spectra")$names$mgf$adduct,
  col_ce = get_params(step = "prepare_libraries_spectra")$names$mgf$collision_energy,
   col_ci = get_params(step = "prepare_libraries_spectra")$names$mgf$compound_id,
   col_em = get_params(step = "prepare_libraries_spectra")$names$mgf$exact_mass,
   col_in = get_params(step = "prepare_libraries_spectra")$names$mgf$inchi,
  col_io = get_params(step = "prepare_libraries_spectra")$names$mgf$inchi_no_stereo,
   col_ik = get_params(step = "prepare_libraries_spectra")$names$mgf$inchikey,
  col_il = get_params(step = "prepare_libraries_spectra")$names$mgf$inchikey_no_stereo,
  col_mf = get_params(step = "prepare_libraries_spectra")$names$mgf$molecular_formula,
   col_na = get_params(step = "prepare_libraries_spectra")$names$mgf$name,
   col_po = get_params(step = "prepare_libraries_spectra")$names$mgf$polarity,
   col_sm = get_params(step = "prepare_libraries_spectra")$names$mgf$smiles,
  col_sn = get_params(step = "prepare_libraries_spectra")$names$mgf$smiles_no_stereo,
   col_si = get_params(step = "prepare_libraries_spectra")$names$mgf$spectrum_id,
   col_sp = get_params(step = "prepare_libraries_spectra")$names$mgf$splash,
   col_sy = get_params(step = "prepare_libraries_spectra")$names$mgf$synonyms,
   col_xl = get_params(step = "prepare_libraries_spectra")$names$mgf$xlogp
)
```

Arguments

`input`	File containing spectra
`nam_lib`	Metadata to identify the library
`col_ad`	Name of the adduct in mgf
`col_ce`	Name of the collision energy in mgf
`col_ci`	Name of the compound id in mgf
`col_em`	Name of the exact mass in mgf
`col_in`	Name of the InChI in mgf
`col_io`	Name of the InChI without stereo in mgf
`col_ik`	Name of the InChIKey in mgf
`col_il`	Name of the InChIKey without stereo in mgf

col_mf	Name of the molecular formula in mgf
col_na	Name of the name in mgf
col_po	Name of the polarity in mgf
col_sm	Name of the SMILES in mgf
col_sn	Name of the SMILES without stereo in mgf
col_si	Name of the spectrum id in mgf
col_sp	Name of the SPLASH in mgf
col_sy	Name of the synonyms in mgf
col_xl	Name of the xlogp in mgf
polarity	MS polarity

Value

The path to the prepared spectral library

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
prepare_libraries_spectra()
unlink("data", recursive = TRUE)

## End(Not run)
```

---

prepare_params *Prepare params*

Description

This function prepares main parameters

Usage

```
prepare_params(
  params_small = get_params(step = "prepare_params"),
  params_advanced = get_params(step = "prepare_params_advanced"),
  step = NA
)
```

Arguments

params_small	params_small
params_advanced	params_advanced
step	Step

Value

The path to the yaml files containing prepared parameters

Examples

```
NULL
```

`prepare_taxa` *Prepare taxa*

Description

This function performs taxon name preparation to match the Open Tree of Life taxonomy

Usage

```
prepare_taxa(
  input = get_params(step = "prepare_taxa")$files$features$raw,
  extension = get_params(step = "prepare_taxa")$names$extension,
  name_features = get_params(step = "prepare_taxa")$names$features,
  name_filename = get_params(step = "prepare_taxa")$names$filename,
  colname = get_params(step = "prepare_taxa")$names$taxon,
  metadata = get_params(step = "prepare_taxa")$files$metadata$raw,
  top_k = get_params(step = "prepare_taxa")$organisms$candidates,
  org_tax_ott = get_params(step =
    "prepare_taxa")$files$libraries$sop$merged$organisms$taxonomies$ott,
  output = get_params(step = "prepare_taxa")$files$metadata$prepared,
  taxon = get_params(step = "prepare_taxa")$organisms$taxon
)
```

Arguments

`input`	File containing your features intensities
`extension`	Does your column names contain the file extension? (MZmine mainly)
`name_features`	Name of the features column in the features file
`name_filename`	Name of the file name column in the metadata file
`colname`	Name of the column containing biological source information
`metadata`	File containing your metadata including biological source
`top_k`	Number of organisms to be retained per feature top intensities
`org_tax_ott`	File containing Open Tree of Life Taxonomy
`output`	Output file
`taxon`	If you want to enforce all features to a given taxon, put its name here.

Details

Depending if the features are aligned between samples originating from various organisms or not, It can either attribute all features to a single organism, or attribute them to multiple ones, according to their relative intensities among the samples.

Value

The path to the prepared taxa

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
dir <- paste0(github, repo)
org_tax_ott <- get_params(step = "prepare_taxa")$files$libraries$sop$merged$organisms$taxonomies$ott |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
get_file(url = paste0(dir, org_tax_ott), export = org_tax_ott)
get_file(
  url = get_default_paths()$urls$examples$features,
  export = get_params(step = "prepare_taxa")$files$features$raw
)
prepare_taxa(
  taxon = "Homo sapiens",
  org_tax_ott = org_tax_ott
)
unlink("data", recursive = TRUE)

## End(Not run)
```

---

pre_harmonize_names_sirius

Pre harmonize names sirius

Description

This function pre harmonizes Sirius names to make them compatible

Usage

```
pre_harmonize_names_sirius(x)
```

Arguments

x	Character string containing a name

Value

Character string with the name modified according to the rules specified in the function

Examples

```
prepared_name <- tima:::pre_harmonize_names_sirius("My name/suffix")
```

---

read_from_sirius_zip *Read from SIRIUS zip*

Description

This function reads files from Sirius compressed workspace

Usage

```
read_from_sirius_zip(sirius_zip, file)
```

Arguments

sirius_zip	Compressed directory containing the Sirius results
file	File to be read

Examples

```
NULL
```

---

replace_id *Replace ID in file paths*

Description

This function replaces the default ID in the example by a user-specified one

Usage

```
replace_id(
  x,
  user_filename = get_params(step = "prepare_params")$files$pattern,
  user_gnps = get_params(step = "prepare_params")$gnps$id,
  example_gnps = get_default_paths()$gnps$example
)
```

Arguments

x	a character string containing the default ID
user_filename	a user-specified value for a file name job ID
user_gnps	a user-specified value for a GNPS job ID
example_gnps	an example value for a GNPS job ID

Value

Character string with the GNPS job ID modified according to the rules specified in the function

Examples

```
tima:::replace_id(
  x = "example/123456_features.tsv",
  user_gnps = NULL,
  user_filename = "Foo"
)
```

---

round_reals *Round reals*

Description

This function rounds some reals in a dataframe

Usage

```
round_reals(df, dig = 5)
```

Arguments

df	Dataframe to use
dig	Number of digits

Examples

```
NULL
```

run_app *Run app*

Description

This function runs the app

Usage

```
run_app(host = "127.0.0.1", port = 3838, browser = TRUE)
```

Arguments

host	Host. Default to 127.0.0.1
port	Port. Default to 3838
browser	Flag for browser use. Default to TRUE

Value

Opens the app

Examples

```
NULL
```

sanitize_spectra *Sanitize spectra*

Description

This function sanitizes spectra

Usage

```
sanitize_spectra(spectra, cutoff = 0, dalton = 0.01, polarity = NA, ppm = 10)
```

Arguments

spectra	Spectra object
cutoff	Absolute minimal intensity
dalton	Dalton tolerance
polarity	Polarity
ppm	PPM tolerance

Value

The sanitized spectra

Examples

```
data.frame(
  FEATURE_ID = c("FT001", "FT002", "FT003"),
  mz = c(list(123.4567, 234.5678, 345.6789))
) |>
  Spectra::Spectra() |>
  sanitize_spectra()
```

`select_annotations_columns` *Select annotations columns*

Description

This function selects annotations columns

Usage

```
select_annotations_columns(
  df,
  str_stereo = get("str_stereo", envir = parent.frame()),
  str_met = get("str_met", envir = parent.frame()),
  str_nam = get("str_nam", envir = parent.frame()),
  str_tax_cla = get("str_tax_cla", envir = parent.frame()),
  str_tax_npc = get("str_tax_npc", envir = parent.frame())
)
```

Arguments

`df`	Dataframe
`str_stereo`	File containing structures stereo
`str_met`	File containing structures metadata
`str_nam`	File containing structures names
`str_tax_cla`	File containing Classyfire taxonomy
`str_tax_npc`	File containing NPClassifier taxonomy

Value

The dataframe with annotation columns selected

Examples

```
NULL
```

`select_sirius_columns_canopus` *Select sirius columns (canopus)*

Description

This function selects sirius columns (canopus)

Usage

```
select_sirius_columns_canopus(df, sirius_version)
```

Arguments

`df`	Dataframe
`sirius_version`	Sirius version

Value

The dataframe with selected canopus columns

Examples

```
NULL
```

`select_sirius_columns_formulas` *Select sirius columns (formulas)*

Description

This function selects sirius columns (formulas)

Usage

```
select_sirius_columns_formulas(df, sirius_version)
```

Arguments

`df`	Dataframe
`sirius_version`	Sirius version

Value

The dataframe with selected sirius columns

Examples

```
NULL
```

`select_sirius_columns_structures` *Select sirius columns (structures)*

Description

This function selects sirius columns (structures)

Usage

```
select_sirius_columns_structures(df, sirius_version)
```

Arguments

`df`	Dataframe
`sirius_version`	Sirius version

Value

The dataframe with selected structure columns

Examples

```
NULL
```

`select_sop_columns` *Select SOP columns*

Description

This function selects sop columns

Usage

```
select_sop_columns(df)
```

Arguments

`df`	Dataframe

Value

The dataframe with selected structure organism pairs columns

Examples

```
NULL
```

`split_tables_sop` *Split Structure Organism Pairs table*

Description

This function splits the structure organism table.

Usage

```
split_tables_sop(table)
```

Arguments

`table`	Table to split

Value

A list of tables from the structure organism pairs tables

Examples

```
NULL
```

`tima_full` *Tima Full*

Description

This function runs everything you need.

Usage

```
tima_full()
```

Value

Everything you need.

Examples

```
NULL
```

transform_score_sirius_csi *Transform score sirius CSI*

Description

This function calculates the mass of M

Usage

```
transform_score_sirius_csi(csi_score, K = 50, scale = 10)
```

Arguments

csi_score	Original CSI score
K	Shift
scale	Scale

Value

A mass

Examples

```
NULL
```

weight_annotations *Weight annotations*

Description

This function weights annotations.

Usage

```
weight_annotations(
  library = get_params(step = "weight_annotations")$files$libraries$sop$merged$keys,
   org_tax_ott = get_params(step =
     "weight_annotations")$files$libraries$sop$merged$organisms$taxonomies$ott,
   str_stereo = get_params(step =
     "weight_annotations")$files$libraries$sop$merged$structures$stereo,
  annotations = get_params(step = "weight_annotations")$files$annotations$filtered,
  canopus = get_params(step = "weight_annotations")$files$annotations$prepared$canopus,
  formula = get_params(step = "weight_annotations")$files$annotations$prepared$formula,
   components = get_params(step =
```

```
  "weight_annotations")$files$networks$spectral$components$prepared,
edges = get_params(step = "weight_annotations")$files$networks$spectral$edges$prepared,
 taxa = get_params(step = "weight_annotations")$files$metadata$prepared,
 output = get_params(step = "weight_annotations")$files$annotations$processed,
candidates_final = get_params(step = "weight_annotations")$annotations$candidates$final,
weight_spectral = get_params(step = "weight_annotations")$weights$global$spectral,
weight_chemical = get_params(step = "weight_annotations")$weights$global$chemical,
weight_biological = get_params(step = "weight_annotations")$weights$global$biological,
 score_biological_domain = get_params(step =
  "weight_annotations")$weights$biological$domain,
 score_biological_kingdom = get_params(step =
  "weight_annotations")$weights$biological$kingdom,
 score_biological_phylum = get_params(step =
  "weight_annotations")$weights$biological$phylum,
 score_biological_class = get_params(step =
  "weight_annotations")$weights$biological$class,
 score_biological_order = get_params(step =
  "weight_annotations")$weights$biological$order,
 score_biological_infraorder = get_params(step =
  "weight_annotations")$weights$biological$infraorder,
 score_biological_family = get_params(step =
  "weight_annotations")$weights$biological$family,
 score_biological_subfamily = get_params(step =
  "weight_annotations")$weights$biological$subfamily,
 score_biological_tribe = get_params(step =
  "weight_annotations")$weights$biological$tribe,
 score_biological_subtribe = get_params(step =
  "weight_annotations")$weights$biological$subtribe,
 score_biological_genus = get_params(step =
  "weight_annotations")$weights$biological$genus,
 score_biological_subgenus = get_params(step =
  "weight_annotations")$weights$biological$subgenus,
 score_biological_species = get_params(step =
  "weight_annotations")$weights$biological$species,
 score_biological_subspecies = get_params(step =
  "weight_annotations")$weights$biological$subspecies,
 score_biological_variety = get_params(step =
  "weight_annotations")$weights$biological$variety,
 score_chemical_cla_kingdom = get_params(step =
  "weight_annotations")$weights$chemical$cla$kingdom,
 score_chemical_cla_superclass = get_params(step =
  "weight_annotations")$weights$chemical$cla$superclass,
 score_chemical_cla_class = get_params(step =
  "weight_annotations")$weights$chemical$cla$class,
 score_chemical_cla_parent = get_params(step =
  "weight_annotations")$weights$chemical$cla$parent,
 score_chemical_npc_pathway = get_params(step =
  "weight_annotations")$weights$chemical$npc$pathway,
```

```
  score_chemical_npc_superclass = get_params(step =
    "weight_annotations")$weights$chemical$npc$superclass,
  score_chemical_npc_class = get_params(step =
    "weight_annotations")$weights$chemical$npc$class,
  minimal_consistency = get_params(step =
    "weight_annotations")$annotations$thresholds$consistency,
  minimal_ms1_bio = get_params(step =
    "weight_annotations")$annotations$thresholds$ms1$biological,
  minimal_ms1_chemo = get_params(step =
    "weight_annotations")$annotations$thresholds$ms1$chemical,
  minimal_ms1_condition = get_params(step =
    "weight_annotations")$annotations$thresholds$ms1$condition,
  ms1_only = get_params(step = "weight_annotations")$annotations$ms1only,
 compounds_names = get_params(step = "weight_annotations")$options$compounds_names,
 high_confidence = get_params(step = "weight_annotations")$options$high_confidence,
  remove_ties = get_params(step = "weight_annotations")$options$remove_ties,
  summarise = get_params(step = "weight_annotations")$options$summarise,
  pattern = get_params(step = "weight_annotations")$files$pattern,
  force = get_params(step = "weight_annotations")$options$force
)
```

Arguments

`library`	Library containing the keys
`org_tax_ott`	File containing organisms taxonomy (OTT)
`str_stereo`	File containing structures stereo
`annotations`	Prepared annotations file
`canopus`	Prepared canopus file
`formula`	Prepared formula file
`components`	Prepared components file
`edges`	Prepared edges file
`taxa`	Prepared taxed features file
`output`	Output file
`candidates_final`	Number of final candidates to keep
`weight_spectral`	Weight for the spectral score
`weight_chemical`	Weight for the biological score
`weight_biological`	Weight for the chemical consistency score
`score_biological_domain`	Score for a `domain` match (should be lower than `kingdom`)
`score_biological_kingdom`	Score for a `kingdom` match (should be lower than `phylum`)

score_biological_phylum
: Score for a phylum match (should be lower than class)

score_biological_class
: Score for a class match (should be lower than order)

score_biological_order
: Score for a order match (should be lower than infraorder)

score_biological_infraorder
: Score for a infraorder match (should be lower than order)

score_biological_family
: Score for a family match (should be lower than subfamily)

score_biological_subfamily
: Score for a subfamily match (should be lower than family)

score_biological_tribe
: Score for a tribe match (should be lower than subtribe)

score_biological_subtribe
: Score for a subtribe match (should be lower than genus)

score_biological_genus
: Score for a genus match (should be lower than subgenus)

score_biological_subgenus
: Score for a subgenus match (should be lower than species)

score_biological_species
: Score for a species match (should be lower than subspecies)

score_biological_subspecies
: Score for a subspecies match (should be lower than variety)

score_biological_variety
: Score for a variety match (should be the highest)

score_chemical_cla_kingdom
: Score for a Classyfire kingdom match (should be lower than Classyfire superclass)

score_chemical_cla_superclass
: Score for a Classyfire superclass match (should be lower than Classyfire class)

score_chemical_cla_class
: Score for a Classyfire class match (should be lower than Classyfire parent)

score_chemical_cla_parent
: Score for a Classyfire parent match (should be the highest)

score_chemical_npc_pathway
: Score for a NPC pathway match (should be lower than NPC superclass)

score_chemical_npc_superclass
: Score for a NPC superclass match (should be lower than NPC class)

score_chemical_npc_class
: Score for a NPC class match (should be the highest)

minimal_consistency
: Minimal consistency score for a class. FLOAT

minimal_ms1_bio
: Minimal biological score to keep MS1 based annotation

`minimal_ms1_chemo`
: Minimal chemical score to keep MS1 based annotation

`minimal_ms1_condition`
: Condition to be used. Must be "OR" or "AND".

`ms1_only`
: Keep only MS1 annotations. BOOLEAN

`compounds_names`
: Report compounds names. Can be very large. BOOLEAN

`high_confidence`
: Report high confidence candidates only. BOOLEAN

`remove_ties`
: Remove ties. BOOLEAN

`summarise`
: Summarize results (1 row per feature). BOOLEAN

`pattern`
: Pattern to identify your job. STRING

`force`
: Force parameters. Use it at your own risk

Value

The path to the weighted annotations

See Also

annotate_masses weight_bio weight_chemo

Examples

```
## Not run:
tima:::copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
dir <- paste0(github, repo)
library <- get_params(step = "weight_annotations")$files$libraries$sop$merged$keys |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
org_tax_ott <- get_params(step = "weight_annotations")$files$libraries$sop$merged$organisms$taxonomies$ott |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
str_stereo <- get_params(step = "weight_annotations")$files$libraries$sop$merged$structures$stereo |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
annotations <- get_params(step = "weight_annotations")$files$annotations$filtered |>
```

```
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
canopus <- get_params(step = "weight_annotations")$files$annotations$prepared$canopus |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
formula <- get_params(step = "weight_annotations")$files$annotations$prepared$formula |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
components <- get_params(step = "weight_annotations")$files$networks$spectral$components$prepared |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
edges <- get_params(step = "weight_annotations")$files$networks$spectral$edges$prepared |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
taxa <- get_params(step = "weight_annotations")$files$metadata$prepared |>
  gsub(
    pattern = ".gz",
    replacement = "",
    fixed = TRUE
  )
get_file(url = paste0(dir, library), export = library)
get_file(url = paste0(dir, org_tax_ott), export = org_tax_ott)
get_file(url = paste0(dir, str_stereo), export = str_stereo)
get_file(url = paste0(dir, annotations), export = annotations)
get_file(url = paste0(dir, canopus), export = canopus)
get_file(url = paste0(dir, formula), export = formula)
get_file(url = paste0(dir, components), export = components)
get_file(url = paste0(dir, edges), export = edges)
get_file(url = paste0(dir, taxa), export = taxa)
weight_annotations(
  library = library,
  org_tax_ott = org_tax_ott,
  str_stereo = str_stereo,
  annotations = annotations,
  canopus = canopus,
  formula = formula,
  components = components,
  edges = edges,
```

```
  taxa = taxa
)
unlink("data", recursive = TRUE)

## End(Not run)
```

`weight_bio` *Weight bio*

Description

This function weights the eventually MS1 complemented annotations according their biological source

Usage

```
weight_bio(
  annotation_table_taxed = get("annotation_table_taxed", envir = parent.frame()),
  structure_organism_pairs_table = get("structure_organism_pairs_table", envir =
    parent.frame()),
  weight_spectral = get("weight_spectral", envir = parent.frame()),
  weight_biological = get("weight_biological", envir = parent.frame()),
  score_biological_domain = get("score_biological_domain", envir = parent.frame()),
  score_biological_kingdom = get("score_biological_kingdom", envir = parent.frame()),
  score_biological_phylum = get("score_biological_phylum", envir = parent.frame()),
  score_biological_class = get("score_biological_class", envir = parent.frame()),
  score_biological_order = get("score_biological_order", envir = parent.frame()),
  score_biological_family = get("score_biological_family", envir = parent.frame()),
  score_biological_tribe = get("score_biological_tribe", envir = parent.frame()),
  score_biological_genus = get("score_biological_genus", envir = parent.frame()),
  score_biological_species = get("score_biological_species", envir = parent.frame()),
  score_biological_variety = get("score_biological_variety", envir = parent.frame())
)
```

Arguments

`annotation_table_taxed`
: Table containing the initial annotation eventually complemented by additional MS1 annotations

`structure_organism_pairs_table`
: Table containing the structure - organism pairs

`weight_spectral`
: Weight for the spectral score

`weight_biological`
: Weight for the biological score

`score_biological_domain`
: Score for a `domain` match (should be lower than `kingdom`)

score_biological_kingdom
: Score for a kingdom match (should be lower than phylum)

score_biological_phylum
: Score for a phylum match (should be lower than class)

score_biological_class
: Score for a class match (should be lower than order)

score_biological_order
: Score for a order match (should be lower than family)

score_biological_family
: Score for a family match (should be lower than tribe)

score_biological_tribe
: Score for a tribe match (should be lower than genus)

score_biological_genus
: Score for a genus match (should be lower than species)

score_biological_species
: Score for a species match (should be lower than variety)

score_biological_variety
: Score for a variety match (should be the highest)

Value

A table containing the biologically weighted annotation

Examples

```
NULL
```

---

weight_chemo *Weight chemo*

Description

This function weights the biologically weighted annotations according their chemical consistency

Usage

```
weight_chemo(
  annot_table_wei_bio_clean = get("annot_table_wei_bio_clean", envir = parent.frame()),
   weight_spectral = get("weight_spectral", envir = parent.frame()),
   weight_biological = get("weight_biological", envir = parent.frame()),
   weight_chemical = get("weight_chemical", envir = parent.frame()),
  score_chemical_cla_kingdom = get("score_chemical_cla_kingdom", envir = parent.frame()),
   score_chemical_cla_superclass = get("score_chemical_cla_superclass", envir =
     parent.frame()),
  score_chemical_cla_class = get("score_chemical_cla_class", envir = parent.frame()),
  score_chemical_cla_parent = get("score_chemical_cla_parent", envir = parent.frame()),
```

```
  score_chemical_npc_pathway = get("score_chemical_npc_pathway", envir = parent.frame()),
   score_chemical_npc_superclass = get("score_chemical_npc_superclass", envir =
     parent.frame()),
  score_chemical_npc_class = get("score_chemical_npc_class", envir = parent.frame())
)
```

Arguments

`annot_table_wei_bio_clean`
: Table containing the biologically weighted annotation

`weight_spectral`
: Weight for the spectral score

`weight_biological`
: Weight for the biological score

`weight_chemical`
: Weight for the chemical consistency score

`score_chemical_cla_kingdom`
: Score for a `Classyfire kingdom` match (should be lower than `Classyfire superclass`)

`score_chemical_cla_superclass`
: Score for a `Classyfire superclass` match (should be lower than `Classyfire class`)

`score_chemical_cla_class`
: Score for a `Classyfire class` match (should be lower than `Classyfire parent`)

`score_chemical_cla_parent`
: Score for a `Classyfire parent` match (should be the highest)

`score_chemical_npc_pathway`
: Score for a `pathway` match (should be lower than `superclass`)

`score_chemical_npc_superclass`
: Score for a `superclass` match (should be lower than `class`)

`score_chemical_npc_class`
: Score for a `class` match (should be the highest)

Value

A table containing the chemically weighted annotation

Examples

```
NULL
```

Index