

Package: tima (via r-universe)

October 19, 2026

Title Taxonomically Informed Metabolite Annotation

Version 2.13.0.9000

Maintainer Adriano Rutz <adafede@gmail.com>

Description TIMA provides a reproducible workflow for taxonomically informed metabolite annotation from feature tables, MS/MS spectra, and optional external resources such as 'SIRIUS' and 'GNPS' outputs. It combines mass, spectral, taxonomic, and structural evidence into a transparent scoring framework that can be inspected step-by-step. The package targets metabolomics practitioners who need configurable, scriptable, and documented annotation pipelines for research and production settings.

License AGPL (>= 3)

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

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

Depends R (>= 4.4.0)

Imports archive (>= 1.1.14), cli (>= 3.6.6), docopt (>= 0.7.2),
fastmatch (>= 1.1.8), fs (>= 2.1.0), httr (>= 1.4.8), httr2 (>= 1.3.0), igraph (>= 2.3.3), jsonlite (>= 2.0.0), lgr (>= 0.5.2),
lifecycle (>= 1.0.5), methods, MetaboCoreUtils (>= 1.18.1),
MsBackendMgf (>= 1.18.0), MsBackendMsp (>= 1.14.0), MsCoreUtils
(>= 1.22.1), msentropy (>= 0.1.4), purrr (>= 1.2.2), R.utils
(>= 2.13.0), reticulate (>= 1.46.0), rlang (>= 1.3.0), rotl (>= 3.1.1), Spectra (>= 1.20.1), stats, targets (>= 1.12.0),
tidyselect (>= 1.2.1), tidyttable (>= 0.11.2), utils, yaml (>= 2.3.12)

Suggests altdoc (>= 0.7.2), BiocManager (>= 1.30.27), BiocVersion (>= 3.22.0), grateful (>= 0.3.0), knitr (>= 1.51), pkgload (>= 1.5.2), quarto (>= 1.5.1), shiny (>= 1.13.0), shinyhelper (>= 0.3.2), shinyjs (>= 2.1.1), shinytest2 (>= 0.5.1),
shinyvalidate (>= 0.1.3), spelling, testthat (>= 3.3.2),
visNetwork (>= 2.1.4), withr (>= 3.0.2)

SystemRequirements python (>= 3.12, < 3.16)

VignetteBuilder quarto

ByteCompile TRUE

Collate 'errors_utils.R' 'validations_utils.R' 'logs_utils.R'
 'validations_params.R' 'safe_fread.R'
 'add_xrefs_to_annotations.R' 'sanitize_data_utils.R'
 'sanitize_data.R' 'create_dir.R' 'export_output.R'
 'get_default_paths.R' 'export_params.R' 'round_reals.R'
 'process_smiles.R' 'columns_utils.R'
 'complement_metadata_structures.R'
 'select_annotations_columns.R' 'constants.R' 'adducts_utils.R'
 'parse_adduct.R' 'mass_evidence_utils.R' 'mass_evidence.R'
 'mass_evidence_hypothesis.R' 'adduct_universe_formula.R'
 'harmonize_adducts.R' 'annotate_masses_features.R'
 'calculate_mass_of_m.R' 'annotate_masses_consistency.R'
 'adduct_universe_matching.R' 'annotate_masses_hypothesis.R'
 'annotate_masses_enforcement.R' 'annotate_masses_coverage.R'
 'parse_yaml_params.R' 'parse_cli_params.R' 'pkg_system_file.R'
 'get_path.R' 'get_params.R' 'dists_utils.R' 'decorate_masses.R'
 'annotate_masses_covariance.R' 'annotate_masses.R'
 'adduct_universe_config.R' 'adduct_universe.R'
 'get_spectra_ids.R' 'sanitize_spectra.R' 'read_mgf_opti.R'
 'import_spectra.R' 'assert_utils.R' 'predicates_utils.R'
 'c_wrappers.R' 'sanitize_spectrum_matrix.R'
 'calculate_similarity.R' 'calculate_entropy_and_similarity.R'
 'annotate_spectra.R' 'benchmark_metrics_utils.R'
 'benchmark_taxize_spectra.R' 'load_yaml_files.R'
 'go_to_cache.R' 'change_params_small.R' 'clean_bio.R'
 'rank_candidates.R' 'clean_chemo_preprocessing.R'
 'clean_chemo_candidates.R' 'summarize_results.R'
 'filter_high_evidence_only.R' 'clean_chemo.R' 'copy_backbone.R'
 'create_components.R' 'create_edges.R' 'create_edges_spectra.R'
 'decorate_bio.R' 'decorate_chemo.R' 'export_spectra_rds.R'
 'weights_utils.R' 'weight_chemo.R' 'extract_chemo_scores.R'
 'extract_spectra.R' 'move_file_safely.R' 'fake_ecmdb.R'
 'fake_hmdb.R' 'fake_lotus.R' 'fake_pubchemlite.R'
 'filter_annotations.R' 'get_compounds_xrefs.R' 'retry_utils.R'
 'get_file.R' 'get_example_sirius.R' 'get_example_files.R'
 'get_gnps_tables.R' 'get_last_version_from_zenodo.R'
 'get_organism_taxonomy_ott.R' 'globals.R'
 'harmonize_names_sirius.R' 'harmonize_spectra.R' 'install.R'
 'mztab_app_utils.R' 'mztab_schema_utils.R' 'mztab_parser.R'
 'mztab_validate.R' 'pre_harmonize_names_sirius.R'
 'prepare_annotations_gnps.R' 'prepare_annotations_mzmine.R'
 'read_mztab.R' 'prepare_annotations_mztab.R'
 'read_from_sirius_zip.R' 'prepare_annotations_sirius_utils.R'
 'select_sirius_columns.R' 'prepare_annotations_sirius.R'
 'prepare_annotations_spectra.R' 'prepare_features_components.R'

```
'prepare_features_edges.R' 'rts_utils.R'
'prepare_features_tables.R' 'prepare_libraries_rt.R'
'select_sop_columns.R' 'prepare_libraries_sop_big.R'
'prepare_libraries_sop_closed.R'
'prepare_libraries_sop_ecmdb.R'
'prepare_libraries_sop_hmdb_like.R'
'prepare_libraries_sop_hmdb.R' 'prepare_libraries_sop_lotus.R'
'split_tables_sop.R' 'prepare_libraries_sop_merged_utils.R'
'prepare_libraries_sop_merged.R'
'prepare_libraries_sop_pubchemlite.R'
'prepare_libraries_spectra.R' 'prepare_params.R'
'prepare_taxa.R' 'replace_id.R' 'run_app.R' 'run_tima.R'
'transform_score_sirius_csi.R' 'weight_bio.R'
'score_combination.R' 'string_utils.R' 'tima-package.R'
'validate_inputs.R' 'weight_annotations_utils.R'
'weight_annotations.R' 'write_mztab.R' 'write_mztab_comments.R'
'write_mztab_merge.R' 'write_mztab_metadata.R'
'write_mztab_tables.R'
```

Config/roxygen2/markdown TRUE

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData TRUE

biocViews metaboliteAnnotation, chemotaxonomy, scoringSystem,
naturalProducts, computationalMetabolomics, taxonomicDistance,
specializedMetabolome

X-schema.org-keywords metaboliteAnnotation, chemotaxonomy,
scoringSystem, naturalProducts, computationalMetabolomics,
taxonomicDistance, specializedMetabolome

Config/pak/sysreqs

cmake libglib-dev make libarchive-dev libpng-dev libuv1-dev libxml2-dev libssl-dev python3

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

Date/Publication 2026-10-19 10:31:00 UTC

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

RemoteRef HEAD

RemoteSha cc9b89da92d332147b3b1925fdb04a79fe823033

Contents

adduct_to_string	5
annotate_masses	6
annotate_spectra	9

build_components_from_edges	11
calculate_mass_of_m	12
calculate_mz_from_mass	14
calculate_similarity	15
change_params_small	16
clean_chemo	18
copy_backbone	21
create_components	22
create_edges	23
create_edges_spectra	24
filter_annotations	26
get_compounds_xrefs	27
get_example_files	28
get_example_sirius	29
get_file	30
get_gnps_tables	31
get_last_version_from_zenodo	32
get_organism_taxonomy_ott	33
go_to_cache	34
harmonize_adducts	36
import_spectra	37
install	38
install_tima	39
parse_adduct	40
prepare_annotations_gnps	41
prepare_annotations_mzmine	42
prepare_annotations_mztab	43
prepare_annotations_sirius	45
prepare_annotations_spectra	47
prepare_features_components	48
prepare_features_edges	50
prepare_features_tables	51
prepare_libraries_rt	52
prepare_libraries_sop_big	54
prepare_libraries_sop_closed	55
prepare_libraries_sop_ecmdb	56
prepare_libraries_sop_hmdb	57
prepare_libraries_sop_lotus	58
prepare_libraries_sop_merged	59
prepare_libraries_sop_pubchemlite	61
prepare_libraries_spectra	62
prepare_params	64
prepare_taxa	65
process_smiles	67
read_mztab	68
run_app	70
run_tima	71
tima_full	72

adduct_to_string 5

transform_score_sirius_csi 73
validate_inputs 75
weight_annotations 76
write_mztab 82

Index 86

adduct_to_string *Build the canonical adduct string from typed components.*

Description

Format (outside-multimer, canonical): [*n*>*M*<-losses><-carrier-losses><+clusters><+carriers>]<|*z*|><sign>

Format (inside-multimer, when *loss_inside_multimer* or *cluster_inside_multimer* is TRUE

and *n_mer* >= 2): [*n*>(*M*<inside-losses><inside-clusters>)<outside-losses><outside-clusters><carriers>]<

Usage

```
adduct_to_string(  
  n_mer,  
  carriers,  
  clusters,  
  losses,  
  z,  
  loss_inside_multimer = FALSE,  
  cluster_inside_multimer = FALSE  
)
```

Arguments

n_mer Integer multimer count.
carriers Named integer vector of carrier counts/signs (e.g. H, Na).
clusters Named integer vector of neutral cluster additions.
losses Named integer vector of neutral losses.
z Integer signed charge.
loss_inside_multimer
 Logical; place losses inside *n*(*M*...) when TRUE.
cluster_inside_multimer
 Logical; place clusters inside *n*(*M*...) when TRUE.

Details

The "inside" variant captures the chemistry where each monomer carries the cluster/loss BEFORE the multimer assembles, e.g. [2(*M*-H₂O)+H]⁺ (two *M*-H₂O monomers dimerize, then protonate) or [2(*M*+NaCl)+H]⁺ (each *M* binds NaCl first, then dimerizes). These have *different* implied neutral masses than their outside-multimer counterparts.

- *n* omitted when *n_mer* == 1
- *|z|* omitted when *|z|* == 1

Value

Canonical adduct string.

Examples

```
## Not run:
# Basic adduct string
adduct_to_string(
  n_mer = 1L,
  carriers = c(H = 1L),
  clusters = integer(),
  losses = integer(),
  z = 1L
)
# Multimeric adduct
adduct_to_string(
  n_mer = 2L,
  carriers = c(H = 1L),
  clusters = c(NaCl = 1L),
  losses = integer(),
  z = 1L
)

## End(Not run)
```

 annotate_masses

 Annotate masses

Description

Mass-based MS1 annotation. The pipeline is a sequence of clearly-bounded steps; each step is documented inline. In short:

1. **Pairs in RT windows.** For every feature, find all other features in the same RT tolerance window (per sample) and compute the m/z delta. The pair is always oriented `(lower\_mz, higher\_mz)` so that $\text{delta} = \text{mz_higher} - \text{mz_lower} \geq 0$.
2. **Adduct edges.** Match each pair's `delta` against the table of precomputed pairwise differences between known mode-specific adducts. A match labels the edge `adduct\_low \_ adduct\_high` and tentatively assigns the corresponding adduct to each endpoint.
3. **Cluster / solvent edges.** Match `delta` against modifier masses (e.g. salts, solvents, ACN, MeOH, Na). A modifier adds mass to the *higher* m/z peak, so the modifier suffix `*<modifier>*` is attached to the *dest* node's adduct hypotheses.

4. **Neutral-loss edges.** Match ``delta`` against neutral-loss masses (e.g. H₂O, CO₂). For an NL pair, the **higher** m/z peak is the precursor and the **lower** m/z peak is the product. The loss suffix ``-<loss>`` is attached to the precursor's adduct hypotheses (so the same neutral M is inferred from both peaks).
5. **Node hypotheses.** Gather, per feature, **all** plausible adduct labels: (a) what we inferred from adduct/cluster/loss edges, (b) any adduct supplied upstream by the preprocessing tool, and (c) the universal baseline ``[M+H]+`` / ``[M-H]-``. Hypotheses are never dropped at this stage.
6. **Library match.** For every ``(feature, candidate_adduct)`` pair, compute the implied neutral mass M and look it up in the library within the ppm tolerance.
7. **Intensity co-variance edges.** For each feature, identify all m/z pairs within the RT window and compute Pearson correlation of intensities across samples ($p < 0.05$). These edges are added to the network alongside rule-based edges, enabling discovery of novel adduct patterns not in the rules.
8. **Network-consensus pruning.** If a feature ends up with several library hits, drop only the candidates whose adduct has **zero** support in the adduct edge graph **and** whose drop still leaves a supported alternative. Ties are kept and drops are logged.
9. **Keep unmatched adducts.** Adduct hypotheses are exported even when no library structure matches, so downstream tools still see the adduct annotation.

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$raw$ms1,
  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$metadata,
  str_nam = NULL,
  str_tax_cla = get_params(step =
    "annotate_masses")$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$clusters,
solvents_list = get_params(step = "annotate_masses")$ms$solvents,
neutral_losses_list = get_params(step = "annotate_masses")$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_dalton = get_params(step = "annotate_masses")$ms$tolerances$mass$dalton$ms1,
tolerance_rt = get_params(step = "annotate_masses")$ms$tolerances$rt$adducts,
correlation_p_threshold = get_params(step =
  "annotate_masses")$ms$tolerances$correlation$p_value
)

```

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	Optional 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
solvents_list	List of solvents 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_dalton	Absolute mass tolerance in Daltons for annotation
tolerance_rt	Tolerance to group adducts. Should be <= 0.05 minutes
correlation_p_threshold	P-value threshold for Pearson correlation of feature intensities. Adduct edges where features show p-value >= correlation_p_threshold are rejected.

Value

Named character of paths to the annotations and edges files.

See Also

Other annotation: [annotate_spectra\(\)](#), [filter_annotations\(\)](#), [weight_annotations\(\)](#), [write_mztab\(\)](#)

Examples

```
## Not run:
annotate_masses()

## End(Not run)
```

annotate_spectra	<i>Annotate spectra</i>
------------------	-------------------------

Description

Annotates MS/MS query spectra against one or more spectral libraries, computing similarity scores and returning best candidate annotations above a similarity threshold.

Usage

```
annotate_spectra(
  input = get_params(step = "annotate_spectra")$files$spectral$raw,
  libraries = get_params(step = "annotate_spectra")$files$libraries$spectral,
  polarity = get_params(step = "annotate_spectra")$ms$polarity,
  output = get_params(step = "annotate_spectra")$files$annotations$raw$spectral$spectral,
  method = get_params(step = "annotate_spectra")$similarities$methods$annotations,
  threshold = get_params(step = "annotate_spectra")$similarities$thresholds$annotations,
  ppm = get_params(step = "annotate_spectra")$ms$tolerances$mass$ppm$ms2,
  dalton = get_params(step = "annotate_spectra")$ms$tolerances$mass$dalton$ms2,
  cutoff = get_params(step = "annotate_spectra")$ms$thresholds$ms2$intensity,
  min_fragments = get_params(step = "annotate_spectra")$ms$thresholds$ms2$min_fragments,
  approx = get_params(step = "annotate_spectra")$annotations$ms2$approx,
  ms1_annotations = NULL,
  qutoff = deprecated()
)
```

Arguments

input	character Vector or list of query spectral file paths (.mgf).
libraries	character Vector or list of library spectral file paths (.mgf / Spectra-supported). Must contain at least one path.
polarity	character MS polarity; one of VALID_MS_MODES ("pos", "neg").
output	character Output file path (the function writes a tabular file here).
method	character Similarity method; one of VALID_SIMILARITY_METHODS.
threshold	numeric Minimal similarity score to retain candidates (0-1).

ppm	numeric Relative mass tolerance (ppm) for MS/MS matching.
dalton	numeric Absolute mass tolerance (Daltons) for MS/MS matching.
cutoff	numeric Intensity cutoff under which MS2 fragments are removed. Non-negative numeric or NULL for dynamic thresholding.
min_fragments	integer Minimum number of fragment peaks a spectrum must have after cleaning to be retained (default: 2).
approx	logical If TRUE perform matching ignoring precursor masses (broader, slower); if FALSE restrict library to precursor-tolerant spectra first.
ms1_annotations	Optional path or data frame containing annotate_masses() output. When provided, query adducts are taken from the MS1 primary assignment (preferably primary + supported_strong) by feature_id, and MGF adduct metadata is used only as fallback.
qutoff	[Deprecated] Use cutoff instead.

Details

This is an orchestration wrapper that performs:

1. Input validation & normalization (query + libraries, numeric params).
2. Query spectra import & light preprocessing (intensity cutoff).
3. Library spectra import, cleaning of empty peak lists, optional polarity filtering, optional precursor-based library size reduction (when approx = FALSE). Libraries are processed one at a time so peak memory stays bounded by a single library instead of the full library set.
4. Similarity computation via calculate_entropy_and_similarity().
5. Candidate metadata extraction (formula, name, etc.).
6. Result shaping: derive error (mz), select canonical output columns, threshold filtering, keep best per (feature_id, library, connectivity layer).
7. Export of parameters & results to the configured output path.

If no annotations are produced (empty inputs or below threshold), a standardized empty template (see fake_annotations_columns()) is exported to ensure downstream code receives expected columns.

Value

Character scalar: the output file path (invisible). Side effect: writes the annotations table to output.

Robustness

The function performs strict validation and logs informative messages. File existence is checked early; similarity computation is wrapped in a tryCatch to surface errors without leaving partially allocated objects.

Performance

Library precursor reduction (when approx = FALSE) limits similarity computation to precursor-tolerant spectra, reducing complexity for large libraries.

See Also

Other annotation: [annotate_masses\(\)](#), [filter_annotations\(\)](#), [weight_annotations\(\)](#), [write_mztab\(\)](#)

Examples

```
## Not run:
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(
  libraries = get_default_paths()$data$source$libraries$spectra$exp$with_rt
)
unlink("data", recursive = TRUE)

## End(Not run)
```

build_components_from_edges

Build components from edges

Description

Public wrapper around the internal community-detection logic used to build components from edge lists.

Usage

```
build_components_from_edges(
  edges,
  name_source = "feature_source",
  name_target = "feature_target",
  resolution = 0.1,
  n_iterations = 2L,
  seed = NULL,
  label_column = "componentindex",
  cut_to_communities = FALSE
)
```

Arguments

edges	A data frame-like object with source/target edge columns and an optional weight column.
name_source	Column name containing source feature identifiers.
name_target	Column name containing target feature identifiers.
resolution	Resolution parameter for weighted community detection.
n_iterations	Number of iterations for the community-detection algorithm.
seed	Random seed for reproducible clustering.
label_column	Column name used for pre-labelled components.
cut_to_communities	Whether to retain only intra-community edges.

Value

A list with the community table, detected method, filtered edges, and component membership.

Examples

```
## Not run:
# Create an edge data frame
edges <- data.frame(
  feature_source = c("f1", "f1", "f2"),
  feature_target = c("f2", "f3", "f3"),
  weight = c(0.8, 0.6, 0.9),
  stringsAsFactors = FALSE
)
result <- build_components_from_edges(edges)
result$components_table

## End(Not run)
```

calculate_mass_of_m	<i>Calculate mass of M</i>
---------------------	----------------------------

Description

This function calculates the neutral mass (M) from an observed m/z value and adduct notation. It accounts for charge, multimers, isotopes, and adduct modifications.

The calculation follows the formula:

$$M = (|z| * (m/z - \text{iso_shift}) - \text{modifications} + z * e_mass) / n_mer$$

where:

- |z| = absolute number of charges
- z = signed charge (``|z| * polarity``)

- m/z = observed mass-to-charge ratio
- iso_shift = `n\_iso \* ISOTOPE\_MASS\_SHIFT\_DALTONS`
- modifications = total neutral mass change from adduct modifications
- e_mass = electron mass
- n_mer = multimer count

Usage

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

Arguments

- mz **numeric** Observed m/z value in Daltons. Must be positive.
- adduct_string **character** Adduct notation string (e.g., [M+H]⁺, [2M+Na]⁺, [M-H₂O+H]⁺)
- electron_mass **numeric** Electron mass in Daltons (default: ELECTRON_MASS_DALTONS from constants.R - CODATA 2018 value)

Value

Numeric neutral mass (M) in Daltons. Returns 0 if: - Adduct parsing fails - Invalid input parameters
- Division by zero would occur (n_mer = 0 or n_charges = 0) Returns NA if calculated mass is negative (physically impossible)

See Also

Other mass-spectrometry: [calculate_mz_from_mass\(\)](#), [calculate_similarity\(\)](#), [harmonize_adducts\(\)](#), [import_spectra\(\)](#), [parse_adduct\(\)](#)

Examples

```
# Simple protonated molecule
calculate_mass_of_m(mz = 123.4567, adduct_string = "[M+H]+")
# Expected: ~122.45 Da

# Sodium adduct
calculate_mass_of_m(mz = 145.4421, adduct_string = "[M+Na]+")
# Expected: ~122.45 Da

# Complex adduct with water loss
calculate_mass_of_m(mz = 105.4467, adduct_string = "[M-H2O+H]+")
# Expected: ~122.45 Da

# Dimer
calculate_mass_of_m(mz = 245.9053, adduct_string = "[2M+H]+")
# Expected: ~122.45 Da

# Doubly charged
calculate_mass_of_m(mz = 62.2311, adduct_string = "[M+2H]2+")
# Expected: ~122.45 Da
```

`calculate_mz_from_mass`*Calculate m/z from neutral mass (inverse operation)*

Description

This is the inverse of `calculate_mass_of_m`. Given a neutral mass and adduct, it calculates the expected m/z value.

Usage

```
calculate_mz_from_mass(  
  neutral_mass,  
  adduct_string,  
  electron_mass = ELECTRON_MASS_DALTONS  
)
```

Arguments

<code>neutral_mass</code>	Numeric neutral mass (M) in Daltons
<code>adduct_string</code>	Character string representing the adduct
<code>electron_mass</code>	Numeric electron mass in Daltons

Value

Numeric m/z value in Daltons

See Also

Other mass-spectrometry: [calculate_mass_of_m\(\)](#), [calculate_similarity\(\)](#), [harmonize_adducts\(\)](#), [import_spectra\(\)](#), [parse_adduct\(\)](#)

Examples

```
# Calculate m/z for a protonated molecule  
calculate_mz_from_mass(neutral_mass = 122.45, adduct_string = "[M+H]+")  
# Expected: ~123.4567  
  
# Verify round-trip calculation  
mass <- 122.45  
adduct <- "[M+H]+"mz <- calculate_mz_from_mass(mass, adduct)  
mass_back <- calculate_mass_of_m(mz, adduct)  
all.equal(mass, mass_back) # Should be TRUE
```

calculate_similarity *Calculate similarity between spectra*

Description

Calculates similarity scores between query and target spectra using either entropy, cosine, or GNPS methods.

****Important:**** For correct results with the GNPS and cosine methods, input spectra should be sanitized (unique, well-separated m/z values; no NaN; sorted by m/z). This is automatically done by [import_spectra()] with ``sanitize = TRUE``.

Usage

```
calculate_similarity(  
  method,  
  query_spectrum,  
  target_spectrum,  
  query_precursor,  
  target_precursor,  
  dalton,  
  ppm,  
  return_matched_peaks = FALSE,  
  ...  
)
```

Arguments

method	character Similarity method: "entropy", "gnps", or "cosine"
query_spectrum	matrix Numeric matrix with columns for mz and intensity
target_spectrum	matrix Numeric matrix with columns for mz and intensity
query_precursor	numeric Precursor m/z value for query
target_precursor	numeric Precursor m/z value for target
dalton	numeric Dalton tolerance for peak matching
ppm	numeric PPM tolerance for peak matching
return_matched_peaks	logical Return matched peaks count? Not compatible with 'entropy' method. Default: FALSE
...	Additional arguments passed to MsCoreUtils::join (cosine only)

Value

Numeric similarity score (0-1), or list with score and matches if `return_matched_peaks = TRUE`.
Returns 0.0 if calculation fails.

See Also

Other mass-spectrometry: [calculate_mass_of_m\(\)](#), [calculate_mz_from_mass\(\)](#), [harmonize_adducts\(\)](#), [import_spectra\(\)](#), [parse_adduct\(\)](#)

Examples

```
sp_1 <- cbind(
  mz = c(10, 36, 63, 91, 93),
  intensity = c(14, 15, 999, 650, 1)
)
precursor_1 <- 123.4567
precursor_2 <- precursor_1 + 14
sp_2 <- cbind(
  mz = c(10, 12, 50, 63, 105),
  intensity = c(35, 5, 16, 999, 450)
)
calculate_similarity(
  method = "entropy",
  query_spectrum = sp_1,
  target_spectrum = sp_2,
  query_precursor = precursor_1,
  target_precursor = precursor_2,
  dalton = 0.005,
  ppm = 10.0
)
calculate_similarity(
  method = "gnps",
  query_spectrum = sp_1,
  target_spectrum = sp_2,
  query_precursor = precursor_1,
  target_precursor = precursor_2,
  dalton = 0.005,
  ppm = 10.0,
  return_matched_peaks = TRUE
)
```

change_params_small *Change Parameters (Convenience Function)*

Description

Updates TIMA workflow parameters for quick setup with a simplified interface. This function modifies the `prepare_params` YAML configuration file by copying provided input files to the appropriate directories and updating parameter values. Implements SOLID principles with clear separation of concerns.

Usage

```
change_params_small(  
  fil_pat = NULL,  
  fil_fea_raw = NULL,  
  fil_met_raw = NULL,  
  fil_sir_raw = NULL,  
  fil_spe_raw = NULL,  
  fil_ann_mzm = NULL,  
  fil_mzt_raw = NULL,  
  ms_pol = NULL,  
  org_tax = NULL,  
  hig_evi = NULL,  
  summarize = NULL,  
  cache_dir = NULL  
)
```

Arguments

fil_pat	Character. Job identifier/pattern for output files (optional)
fil_fea_raw	Character. Path to features file (e.g., from mzmine/SIRIUS)
fil_met_raw	Character. Path to metadata file (optional if single taxon)
fil_sir_raw	Character. Path to SIRIUS annotations directory/zip
fil_spe_raw	Character. Path to spectra file (MGF format with MS1/MS2)
fil_ann_mzm	Character. Path to mzmine annotations file
fil_mzt_raw	Character. Path to an mzTab-M file to import/merge
ms_pol	Character. MS polarity: "pos" or "neg"
org_tax	Character. Scientific name for single-taxon experiments
hig_evi	Logical. Filter for high evidence candidates only
summarize	Logical. Summarize all candidates per feature to single row
cache_dir	Character. Cache directory path (for testing; uses go_to_cache() if NULL)

Details

This function:

- Validates all input files exist before copying
- Copies files to standardized cache locations
- Updates the prepare_params YAML configuration
- Handles NA values properly for YAML null representation

Value

Invisible NULL. Modifies prepare_params YAML as side effect.

See Also

Other workflow: `create_components()`, `create_edges()`, `create_edges_spectra()`, `go_to_cache()`, `install()`, `install_tima()`, `run_app()`, `run_tima()`, `tima_full()`, `validate_inputs()`

Examples

```
## Not run:
# Setup complete workflow parameters
copy_backbone()
change_params_small(
  fil_pat = "gentiana_experiment",
  fil_fea_raw = "data/raw/features.csv",
  fil_met_raw = "data/raw/metadata.tsv",
  fil_sir_raw = "data/raw/sirius_output.zip",
  fil_spe_raw = "data/raw/spectra.mgf",
  fil_ann_mzm = "data/raw/mzmine_annotations.csv",
  fil_mzt_raw = "data/raw/annotations.mztab",
  ms_pol = "pos",
  org_tax = "Gentiana lutea",
  hig_evi = TRUE,
  summarize = FALSE
)

## End(Not run)
```

clean_chemo

Clean Chemical Annotations

Description

Cleans and filters chemically weighted annotation results through a multi-tier pipeline. Applies MS1 score thresholds, percentile filtering, ranking, and optional high-evidence filtering. Ranking is score-based, and optional cluster-consensus promotion can retain plausible siblings when they are the best coherent choices. Returns three-tier output: full (comprehensive), filtered (top candidates), and mini (one row per feature).

Usage

```
clean_chemo(
  annot_table_wei_chemo,
  components_table,
  features_table,
  structure_organism_pairs_table,
  candidates_final,
  best_percentile,
  minimal_ms1_bio,
  minimal_ms1_chemo,
  minimal_ms1_condition,
```

```

    compounds_names,
    high_evidence,
    remove_ties,
    summarize,
    score_chemical_cla_kingdom = 0.2,
    score_chemical_cla_superclass = 0.4,
    score_chemical_cla_class = 0.6,
    score_chemical_cla_parent = 0.8,
    score_chemical_npc_pathway = 0.25,
    score_chemical_npc_superclass = 0.5,
    score_chemical_npc_class = 0.75,
    max_per_score = 7L,
    xrefs_table = NULL,
    enforce_cluster_consensus = TRUE
)

```

Arguments

annot_table_wei_chemo
Data frame with chemically weighted annotations. Required columns: feature_id, candidate_structure_inchikey_connectivity_layer, score_weighted_chemo, score_biological, score_chemical, candidate_score_pseudo_initial

components_table
Data frame with molecular network component assignments. Required columns: feature_id, component_id

features_table Data frame with feature metadata (RT, m/z, etc.). Required columns: feature_id

structure_organism_pairs_table
Data frame linking structures to organisms. Required columns: structure_inchikey_connectivity_layer

candidates_final
Integer, number of top candidates to retain per feature (≥ 1)

best_percentile
Numeric (0-1), percentile threshold for score filtering. Candidates with scores $\geq$ percentile * max_score are kept. Default: 0.9 (90th percentile)

minimal_ms1_bio
Numeric (0-1), minimum biological score for MS1-only annotations

minimal_ms1_chemo
Numeric (0-1), minimum chemical score for MS1-only annotations

minimal_ms1_condition
Character, logical operator for MS1 filtering: "OR" or "AND". "OR" = keep if bio $\geq$ threshold OR chem $\geq$ threshold. "AND" = keep if bio $\geq$ threshold AND chem $\geq$ threshold

compounds_names
Logical, include compound names in output (may increase size)

high_evidence Logical, apply strict high-evidence filters

remove_ties Logical, remove tied scores (keep only highest-ranked)

summarize Logical, collapse results to one row per feature

`score_chemical_cla_kingdom`
 Numeric (0-1), score for ClassyFire kingdom level
`score_chemical_cla_superclass`
 Numeric (0-1), score for ClassyFire superclass level
`score_chemical_cla_class`
 Numeric (0-1), score for ClassyFire class level
`score_chemical_cla_parent`
 Numeric (0-1), score for ClassyFire direct parent level
`score_chemical_npc_pathway`
 Numeric (0-1), score for NPClassifier pathway level
`score_chemical_npc_superclass`
 Numeric (0-1), score for NPClassifier superclass level
`score_chemical_npc_class`
 Numeric (0-1), score for NPClassifier class level
`max_per_score` Integer, max candidates to keep per feature per score. If more exist, they are randomly sampled and a note is added. Default 7.
`xrefs_table` Optional data frame with columns inchikey/prefix/id from `get_compounds_xrefs()`, used to add `candidate_structure_id_*` columns before summarization.
`enforce_cluster_consensus`
 Logical, apply cluster-consensus promotion before candidate sampling and percentile filtering. Default TRUE.

Value

Named list with three data frames:

full All annotations (optionally high-evidence filtered)

filtered Top candidates meeting percentile + rank thresholds

mini One row per feature with best compound/taxonomy

See Also

[weight_chemo](#), [filter_high_evidence_only](#), [summarize_results](#)

Examples

```
## Not run:
results <- clean_chemo(
  annot_table_wei_chemo = annotations,
  features_table = features,
  components_table = components,
  structure_organism_pairs_table = sop_table,
  candidates_final = 10,
  best_percentile = 0.9,
  minimal_ms1_bio = 0.5,
  minimal_ms1_chemo = 0.5,
  minimal_ms1_condition = "OR",
  compounds_names = TRUE,
```

```

    high_evidence = FALSE,
    remove_ties = FALSE,
    summarize = FALSE
)

## End(Not run)

```

copy_backbone

Copy backbone

Description

This function copies the package backbone (default directory structure, configuration files, and parameters) to a cache directory. This sets up the working environment for TIMA workflows.

Usage

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

Arguments

cache_dir	Character string path to the cache directory (default: "~/tima" in user's home directory)
package	Character string name of the package (default: "tima")

Value

NULL (invisibly). Creates cache directory structure as side effect.

Examples

```

## Not run:
# Copy to default cache location
copy_backbone()

# Copy to custom location
copy_backbone(cache_dir = "~/my_tima_cache")

## End(Not run)

```

create_components	<i>Create components</i>
-------------------	--------------------------

Description

This function creates network components (connected subgraphs) from edge lists using igraph. Each component represents a set of features that are connected through spectral similarity or other relationships.

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	Character vector of file path(s) containing edge data. Files should have feature_source and feature_target columns.
output	Character string path for the output components file

Value

Character string path to the created components file

See Also

Other workflow: [change_params_small\(\)](#), [create_edges\(\)](#), [create_edges_spectra\(\)](#), [go_to_cache\(\)](#), [install\(\)](#), [install_tima\(\)](#), [run_app\(\)](#), [run_tima\(\)](#), [tima_full\(\)](#), [validate_inputs\(\)](#)

Examples

```
## Not run:
copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
data_interim <- "data/interim/"
dir <- paste0(github, repo, 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_edges	<i>Create spectral similarity network edges</i>
--------------	---

Description

Calculates pairwise spectral similarity between all spectra to create a network edge list.

Usage

```
create_edges(
  frags,
  nspecs,
  precs,
  method,
  ms2_tolerance,
  ppm_tolerance,
  threshold = NULL,
  matched_peaks = NULL,
  assume_sanitized = FALSE
)
```

Arguments

frags	List of aligned fragment spectra matrices
nspecs	Integer number of spectra
precs	Numeric vector of precursor m/z values
method	Similarity method ("entropy", "gnps", or "cosine")
ms2_tolerance	MS2 tolerance in Daltons
ppm_tolerance	PPM tolerance
threshold	Minimum similarity score threshold. Set to NULL to disable similarity thresholding and retain all pairwise comparisons.
matched_peaks	Minimum number of matched peaks required. Set to NULL to disable matched-peak filtering.
assume_sanitized	Boolean flag to assume if sanitize or not. FALSE

Value

Data frame with columns: feature_id, target_id, score, matched_peaks. Returns empty data frame with NA values if no edges pass thresholds.

See Also

Other workflow: [change_params_small\(\)](#), [create_components\(\)](#), [create_edges_spectra\(\)](#), [go_to_cache\(\)](#), [install\(\)](#), [install_tima\(\)](#), [run_app\(\)](#), [run_tima\(\)](#), [tima_full\(\)](#), [validate_inputs\(\)](#)

Examples

```
## Not run:
edges <- create_edges(
  frags = fragment_list,
  nspecs = length(fragment_list),
  precs = precursor_mz,
  method = "gnps",
  ms2_tolerance = 0.02,
  ppm_tolerance = 10,
  threshold = 0.7,
  matched_peaks = 6
)

## End(Not run)
```

create_edges_spectra *Create edges spectra*

Description

This function creates molecular network edges based on MS2 fragmentation spectra similarity. Compares all spectra against each other using spectral similarity metrics to identify related features. It annotates the resulting edges with community labels derived from the similarity network and retains only intra-community edges, so the edge file itself carries the community structure.

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$spectral,
  name_source = get_params(step = "create_edges_spectra")$names$source,
  name_target = get_params(step = "create_edges_spectra")$names$target,
  method = get_params(step = "create_edges_spectra")$similarities$methods$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,
  cutoff = get_params(step = "create_edges_spectra")$ms$thresholds$ms2$intensity,
  min_fragments = get_params(step =
    "create_edges_spectra")$ms$thresholds$ms2$min_fragments,
  resolution = 0.1,
  n_iterations = 2L,
  seed = NULL
)
```

Arguments

input	character Path or list of paths to query MGF file(s) containing spectra
output	character Path for output edges file
name_source	character Name of source feature column
name_target	character Name of target feature column
method	character Similarity method to use
ppm	numeric Relative mass tolerance in ppm
dalton	numeric Absolute mass tolerance in Daltons
cutoff	numeric Intensity cutoff below which MS2 fragments are removed. Non-negative numeric or NULL for dynamic thresholding.
min_fragments	integer Minimum number of fragment peaks a spectrum must have after cleaning to be retained
resolution	numeric Resolution parameter for the Leiden/Louvain algorithm (higher values -> more, smaller communities). Default: 0.1.
n_iterations	integer Number of iterations for the Leiden algorithm. Ignored for Louvain. Default: 2.
seed	integer Random seed for reproducible clustering. If NULL, a random seed is used. Default: NULL.

Value

Character string path to the created spectral edges file.

See Also

Other workflow: [change_params_small\(\)](#), [create_components\(\)](#), [create_edges\(\)](#), [go_to_cache\(\)](#), [install\(\)](#), [install_tima\(\)](#), [run_app\(\)](#), [run_tima\(\)](#), [tima_full\(\)](#), [validate_inputs\(\)](#)

Examples

```
## Not run:
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)
```

filter_annotations	<i>Filter annotations</i>
--------------------	---------------------------

Description

This function filters initial annotations by removing MS1-only annotations that also have quality spectral matches (gated on similarity and matched peaks), and joins retention time library data when available. RT deltas are computed but no hard cutoff is applied; the downstream scoring system uses a sigmoid penalty to handle RT deviations gracefully.

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	Character vector or list of paths to prepared annotation files
features	Character string path to prepared features file. Must contain a feature_id column. The rt column is optional; if absent, RT filtering is skipped even when an RT library is provided.
rts	Character string path to prepared retention time library (optional)
output	Character string path for filtered annotations output
tolerance_rt	Numeric RT tolerance in minutes (used for deduplication of multiple RT library matches; no hard cutoff is applied)

Value

Character string path to the filtered annotations file

See Also

Other annotation: [annotate_masses\(\)](#), [annotate_spectra\(\)](#), [weight_annotations\(\)](#), [write_mztab\(\)](#)

Examples

```
## Not run:
copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
```

```

dir <- paste0(github, repo)
ann <- get_params(step =
  "filter_annotations")$files$annotations$prepared$structural[[2L]] |>
  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 = ann,
  features = features,
  rts = rts
)
unlink("data", recursive = TRUE)

## End(Not run)

```

get_compounds_xrefs	<i>Retrieve external database identifiers for compounds by InChIKey via Wikidata</i>
---------------------	--

Description

Fetches mappings from the Bioregistry for a set of Wikidata property IDs, queries QLever for compound identifiers, and returns a tidy long data.frame with one row per InChIKey × database combination, including Wikidata QIDs. Results are cached to disk; the query is only re-run when the cached file is older than max_age_hours (default 24 h) or does not exist.

Usage

```

get_compounds_xrefs(
  props = c("P231", "P592", "P661", "P662", "P665", "P683", "P715", "P2057", "P2063",
    "P2877", "P8691"),
  bioregistry_url = paste0("https://raw.githubusercontent.com/",
    "biopragnatics/bioregistry/refs/heads/main/",
    "src/bioregistry/data/bioregistry.json"),
  qlever_url = "https://qllever.cs.uni-freiburg.de/api/wikidata",
  max_age_hours = 24,
  output = get_default_paths()$data$interim$xrefs$compounds
)

```

Arguments

props	Character vector of Wikidata property IDs (without wdt: prefix), e.g. c("P683", "P592").
bioregistry_url	URL to the bulk bioregistry JSON. Defaults to the canonical GitHub raw URL.
qllever_url	QLever SPARQL endpoint URL.
max_age_hours	Numeric maximum age (in hours) of the cached file before it is refreshed. Default 24.
output	Character file path for the cached result. When used inside a targets pipeline with format = "file", this path is tracked automatically.

Value

Character path to the exported file (invisibly), for **targets** format = "file" compatibility.

See Also

Other data-retrieval: [get_example_files\(\)](#), [get_file\(\)](#), [get_gnps_tables\(\)](#), [get_last_version_from_zenodo\(\)](#), [get_organism_taxonomy_ott\(\)](#)

Examples

```
## Not run:
props <- c("P231", "P592", "P683", "P715")
result_path <- get_compounds_xrefs(props)
utils::head(tidytable::fread(result_path))

## End(Not run)
```

get_example_files	<i>Get example files</i>
-------------------	--------------------------

Description

This function downloads example data files for testing and demonstration purposes. Supports downloading features, metadata, SIRIUS annotations, mass spectra, and spectral libraries with retention times.

Usage

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

Arguments

example	Character vector specifying which example files to download. Valid options: "features", "metadata", "sirius", "spectra", "spectral_lib_with_rt"
in_cache	Logical whether to store files in the cache directory (default: TRUE)

Value

NULL (invisibly). Downloads files as a side effect.

See Also

Other data-retrieval: [get_compounds_xrefs\(\)](#), [get_file\(\)](#), [get_gnps_tables\(\)](#), [get_last_version_from_zenodo\(\)](#), [get_organism_taxonomy_ott\(\)](#)

Examples

```
## Not run:
# Download features and metadata examples
get_example_files(example = c("features", "metadata"))

# Download all example files to cache
get_example_files(
  example = c("features", "metadata", "sirius", "spectra"),
  in_cache = TRUE
)

## End(Not run)
```

get_example_sirius	<i>Get example sirius</i>
--------------------	---------------------------

Description

This function downloads example SIRIUS annotation files for testing and demonstration purposes. Downloads both SIRIUS v5 and v6 format files.

Usage

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

Arguments

url	list List containing URLs for SIRIUS examples (must have \$v5 and \$v6 elements)
export	list List containing export paths for SIRIUS examples (must have \$v5 and \$v6 elements)

Value

NULL (invisibly). Downloads files as a side effect.

Examples

```
## Not run:  
get_example_sirius()  
  
## End(Not run)
```

get_file

Download file from URL

Description

Downloads a file from a URL with robust error handling, retry logic, and validation. Automatically creates necessary directories and validates downloaded content. Skips download if file already exists.

Usage

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

Arguments

url	character URL of the file to download
export	character File path where the file should be saved
limit	integer Timeout limit in seconds (default: 3600 = 1 hour)

Value

Path to the downloaded file (invisibly)

See Also

Other data-retrieval: [get_compounds_xrefs\(\)](#), [get_example_files\(\)](#), [get_gnps_tables\(\)](#), [get_last_version_from_z](#), [get_organism_taxonomy_ott\(\)](#)

Examples

```
## Not run:  
get_file(  
  url = "https://example.com/data.tsv",  
  export = "data/source/data.tsv"  
)  
  
## End(Not run)
```

get_gnps_tables	<i>Get GNPS Tables</i>
-----------------	------------------------

Description

This function downloads and retrieves GNPS (Global Natural Products Social Molecular Networking) result tables from a completed job. It fetches features, metadata, spectra, and annotation files from GNPS servers. When a job ID is not provided or GNPS resources are missing, small fake files are written so downstream steps do not fail during testing.

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	Character string GNPS job ID (32 characters). Can be NULL or empty string to skip download.
gnps_job_example	Character string example GNPS job ID for testing
filename	Character string name of the file to download (used for fake outputs)
workflow	Character string indicating workflow type: "fbmn" (feature-based) or "classical" molecular networking
path_features	Character string path for features output (file path)
path_metadata	Character string path for metadata output (file path or list)
path_spectra	Character string path for spectra output (file path)
path_source	Character string path to store source files
path_interim_a	Character string path to store interim annotations
path_interim_f	Character string path to store interim features

Value

A named character vector with paths to the written/available files.

See Also

Other data-retrieval: [get_compounds_xrefs\(\)](#), [get_example_files\(\)](#), [get_file\(\)](#), [get_last_version_from_zenodo\(\)](#), [get_organism_taxonomy_ott\(\)](#)

Examples

```
## Not run:
# Download GNPS FBMN results
paths <- get_gnps_tables(
  gnps_job_id = "1234567890abcdef",
  workflow = "fbmn",
  path_features = "data/interim/features/features.tsv",
  path_metadata = "data/source/metadata.tsv",
  path_spectra = "data/interim/annotations/spectra.mgf"
)

# Access downloaded files
features <- read.delim(paths["features"])

## End(Not run)
```

```
get_last_version_from_zenodo
```

Get Latest Version from Zenodo

Description

Retrieves the latest version of a file from a Zenodo repository record. This function checks the file size and only downloads if the local file is missing or differs from the remote version. Implements robust error handling and retry logic.

Usage

```
get_last_version_from_zenodo(doi, pattern, path, timeout_s = 90)
```

Arguments

doi	Character. Zenodo DOI (e.g., "10.5281/zenodo.5794106")
pattern	Character. Pattern to identify the specific file to download
path	Character. Local path where the file should be saved
timeout_s	Numeric. Metadata request timeout in seconds (default: 90)

Details

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

This function:

- Validates DOI format and input parameters
- Fetches the latest version metadata from Zenodo API
- Finds files matching the specified pattern
- Compares local and remote file sizes to avoid unnecessary downloads
- Downloads only if needed, with retry logic
- Creates necessary directories automatically

Value

Character path to the downloaded (or existing) file

See Also

Other data-retrieval: [get_compounds_xrefs\(\)](#), [get_example_files\(\)](#), [get_file\(\)](#), [get_gnps_tables\(\)](#), [get_organism_taxonomy_ott\(\)](#)

Examples

```
## Not run:
# Download LOTUS database from Zenodo
get_last_version_from_zenodo(
  doi = "10.5281/zenodo.5794106",
  pattern = "lotus.csv.gz",
  path = "data/source/libraries/sop/lotus.csv.gz"
)

# The function will skip download if file exists with correct size
get_last_version_from_zenodo(
  doi = "10.5281/zenodo.5794106",
  pattern = "lotus.csv.gz",
  path = "data/source/libraries/sop/lotus.csv.gz"
)

## End(Not run)
```

get_organism_taxonomy_ott

Get organism taxonomy (Open Tree of Life Taxonomy)

Description

This function retrieves taxonomic information from the Open Tree of Life (OTT) taxonomy service. It cleans organism names, queries the OTT API, and returns structured taxonomic data including OTT IDs and hierarchical classifications.

Usage

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

Arguments

df	data.frame Data frame containing organism names in a column named "organism"
url	character Character string URL of the OTT API endpoint (default: production API, can be changed for testing)
retry	logical Logical indicating whether to retry failed queries using only the generic epithet (genus name) when full species names fail (default: TRUE)

Value

Data frame with taxonomic information including OTT IDs, ranks, and taxonomic hierarchy. Returns empty template if API is unavailable.

See Also

Other data-retrieval: [get_compounds_xrefs\(\)](#), [get_example_files\(\)](#), [get_file\(\)](#), [get_gnps_tables\(\)](#), [get_last_version_from_zenodo\(\)](#)

Examples

```
## Not run:
# Single organism
df <- data.frame(organism = "Homo sapiens")
taxonomy <- get_organism_taxonomy_ott(df)

# Multiple organisms
df <- data.frame(organism = c("Homo sapiens", "Arabidopsis thaliana"))
taxonomy <- get_organism_taxonomy_ott(df)

## End(Not run)
```

go_to_cache

Navigate to cache directory

Description

Creates and navigates to a cache directory in the user's home directory. Useful for storing temporary files, intermediate results, and downloaded data in a consistent location across sessions.

Usage

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

Arguments

dir [character](#) Character string name of cache directory (default: ".tima"). Created in user's home directory. Must be non-empty.

Details

The function:

- Constructs full path in user's home directory
- Creates directory if it doesn't exist
- Changes working directory to cache location
- Logs all operations

Cache directory persists across R sessions until explicitly deleted.

Value

Path to cache directory (invisibly). Changes working directory as side effect.

See Also

Other workflow: [change_params_small\(\)](#), [create_components\(\)](#), [create_edges\(\)](#), [create_edges_spectra\(\)](#), [install\(\)](#), [install_tima\(\)](#), [run_app\(\)](#), [run_tima\(\)](#), [tima_full\(\)](#), [validate_inputs\(\)](#)

Examples

```
## Not run:
# Default cache (~/.tima)
go_to_cache()

# Custom cache
go_to_cache(dir = ".my_cache")

# Store path
cache_path <- go_to_cache()

## End(Not run)
```

harmonize_adducts	<i>Harmonize adduct notations</i>
-------------------	-----------------------------------

Description

Standardizes adduct notations in a dataframe by replacing various forms with canonical representations. Uses a translation table for efficient batch replacement.

Usage

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

Arguments

df	Data frame or tibble containing adduct column
adducts_colname	Character string name of the adduct column (default: "adduct")
adducts_translations	Named character vector mapping original adduct notations (names) to standardized forms (values). If missing, returns dataframe unchanged.

Details

Common adduct variations like "M+H", "[M+H]", and "(M+H)+" are standardized to a consistent format (e.g., "[M+H]+"). This ensures compatibility across different MS tools and databases.

Value

Data frame with harmonized adduct column

See Also

Other mass-spectrometry: [calculate_mass_of_m\(\)](#), [calculate_mz_from_mass\(\)](#), [calculate_similarity\(\)](#), [import_spectra\(\)](#), [parse_adduct\(\)](#)

Examples

```
## Not run:
df <- data.frame(adduct = c("M+H", "[M+Na]+", "(M-H)-"))
translations <- c("M+H" = "[M+H]+", "(M-H)-" = "[M-H]-")
harmonize_adducts(df, adducts_translations = translations)

## End(Not run)
```

import_spectra	<i>Import spectra</i>
----------------	-----------------------

Description

This function imports mass spectra from various file formats (.mgf, .msp, .rds), harmonizes meta-data field names, filters by MS level and polarity, optionally combines replicate spectra, and sanitizes peak data.

Usage

```
import_spectra(  
  file,  
  cutoff = NULL,  
  dalton = 0.01,  
  min_fragments = 1L,  
  polarity = NA,  
  ppm = 10,  
  sanitize = TRUE,  
  combine = TRUE  
)
```

Arguments

file	Character string path to the spectrum file (.mgf, .msp, or .rds)
cutoff	Numeric absolute minimal intensity threshold (default: NULL)
dalton	Numeric Dalton tolerance for peak matching (default: 0.01)
min_fragments	Integer minimum number of fragment peaks required to keep a spectrum after sanitization (default: 1)
polarity	Character string for polarity filtering: "pos", "neg", or NA to keep all (default: NA)
ppm	Numeric PPM tolerance for peak matching (default: 10)
sanitize	Logical flag indicating whether to sanitize spectra (default: TRUE)
combine	Logical flag indicating whether to combine replicate spectra (default: TRUE)

Value

Spectra object containing the imported and processed spectra

See Also

Other mass-spectrometry: [calculate_mass_of_m\(\)](#), [calculate_mz_from_mass\(\)](#), [calculate_similarity\(\)](#), [harmonize_adducts\(\)](#), [parse_adduct\(\)](#)

Examples

```
## Not run:
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
)

## End(Not run)
```

install

Install TIMA Package and Dependencies (DEPRECATED)

Description

[Deprecated]

DEPRECATED: Use [install_tima\(\)](#) instead. `install()` will be removed in a future version. The generic name `install` risks masking other packages.

Usage

```
install(
  package = "tima",
  repos = c("https://taxonomicallyinformedannotation.r-universe.dev",
    "https://cran.r-universe.dev", "https://bioc.r-universe.dev"),
  type = getOption("pkgType"),
  dependencies = TRUE
)
```

Arguments

package	character Name of the package (default: "tima")
repos	character Vector of repository URLs
type	character Type of install ("source", "binary")
dependencies	logical Whether to install dependencies (default: TRUE)

Value

NULL (invisibly).

See Also

Other workflow: [change_params_small\(\)](#), [create_components\(\)](#), [create_edges\(\)](#), [create_edges_spectra\(\)](#), [go_to_cache\(\)](#), [install_tima\(\)](#), [run_app\(\)](#), [run_tima\(\)](#), [tima_full\(\)](#), [validate_inputs\(\)](#)

Examples

```
## Not run:
# DEPRECATED – use install_tima() instead
install_tima()

## End(Not run)
```

install_tima

*Install TIMA Package and Dependencies***Description**

Installs or updates the TIMA package from r-universe and sets up a Python virtual environment with dependencies.

Usage

```
install_tima(
  package = "tima",
  repos = c("https://taxonomicallyinformedannotation.r-universe.dev",
            "https://cran.r-universe.dev", "https://bioc.r-universe.dev"),
  type = getOption("pkgType"),
  dependencies = TRUE
)
```

Arguments

package	character Name of the package (default: "tima")
repos	character Vector of repository URLs
type	character Type of install ("source", "binary")
dependencies	logical Whether to install dependencies (default: TRUE)

Value

NULL (invisibly). Installs packages and sets up Python environment as side effects.

See Also

Other workflow: [change_params_small\(\)](#), [create_components\(\)](#), [create_edges\(\)](#), [create_edges_spectra\(\)](#), [go_to_cache\(\)](#), [install\(\)](#), [run_app\(\)](#), [run_tima\(\)](#), [tima_full\(\)](#), [validate_inputs\(\)](#)

Examples

```
## Not run:
install_tima()

## End(Not run)
```

parse_adduct	<i>Parse adduct</i>
--------------	---------------------

Description

This function parses mass spectrometry adduct notation strings into their components: multimer count, isotope shift, modifications, charge state, and charge sign. It handles complex adducts with multiple additions/losses.

Usage

```
parse_adduct(adduct_string, regex = ADDUCT_REGEX_PATTERN)
```

Arguments

adduct_string [character](#) Character string representing the adduct in standard notation (e.g., "[M+H]+", "[2M+Na]+", "[M-H2O+H]+")

regex [character](#) Character string regular expression pattern for parsing (default: uses ADDUCT_REGEX_PATTERN from constants)

Value

Named numeric vector containing:

n_mer	Integer number of monomers (e.g., 2 for dimer, 1 for monomer)
n_iso	Integer isotope shift (e.g., 1 for M+1 isotopologue, 0 for monoisotopic)
los_add_clu	Numeric total mass change in Daltons from all modifications
n_charges	Integer absolute number of charges (always positive)
charge	Integer charge polarity (+1 for positive mode, -1 for negative mode)

Returns all zeros if parsing fails.

See Also

Other mass-spectrometry: [calculate_mass_of_m\(\)](#), [calculate_mz_from_mass\(\)](#), [calculate_similarity\(\)](#), [harmonize_adducts\(\)](#), [import_spectra\(\)](#)

Examples

```
# Simple adducts
parse_adduct("[M+H]+") # Protonated molecule
parse_adduct("[M-H]-") # Deprotonated molecule
parse_adduct("[M+Na]+") # Sodium adduct

# Complex adducts
parse_adduct("[2M+Na]+") # Dimer with sodium
parse_adduct("[M-H2O+H]+") # Protonated with water loss
```

```
## Not run:
# Advanced cases
parse_adduct("[M1+H]+") # M+1 isotopologue
parse_adduct("[2M1-C6H12O6 (hexose)+NaCl+H]2+") # Complex modification

## End(Not run)
```

```
prepare_annotations_gnps
```

Prepare annotations GNPS

Description

This function prepares GNPS spectral library matching results by standardizing column names, integrating structure metadata, and formatting for downstream TIMA annotation workflows.

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_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	character Character string or vector of paths to GNPS annotation files
output	character Character string path for prepared GNPS annotations output
str_stereo	character Character string path to structures stereochemistry file
str_met	character Character string path to structures metadata file
str_tax_cla	character Character string path to ClassyFire taxonomy file
str_tax_npc	character Character string path to NPClassifier taxonomy file

Value

Character string path to prepared GNPS annotations

See Also

Other preparation: `prepare_annotations_mzmine()`, `prepare_annotations_mztab()`, `prepare_annotations_sirius()`, `prepare_annotations_spectra()`, `prepare_features_components()`, `prepare_features_edges()`, `prepare_features_tables()`, `prepare_libraries_rt()`, `prepare_libraries_sop_big()`, `prepare_libraries_sop_ecmdb()`, `prepare_libraries_sop_hmdb()`, `prepare_libraries_sop_lotus()`, `prepare_libraries_sop_merged()`, `prepare_libraries_sop_pubchemlite()`, `prepare_libraries_spectra()`, `prepare_params()`, `prepare_taxa()`, `read_mztab()`

Examples

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

## End(Not run)
```

```
prepare_annotations_mzmine
```

Prepare annotations mzmine

Description

This function prepares mzmine spectral library matching results by standardizing column names, integrating structure metadata, and formatting for downstream TIMA annotation workflows.

Usage

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

Arguments

input	character Character string or vector of paths to mzmine annotation files
output	character Character string path for prepared mzmine annotations output
str_stereo	character Character string path to structures stereochemistry file
str_met	character Character string path to structures metadata file
str_tax_cla	character Character string path to ClassyFire taxonomy file
str_tax_npc	character Character string path to NPClassifier taxonomy file

Value

Character string path to prepared mzmine annotations

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_bigg\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

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

## End(Not run)
```

prepare_annotations_mztab

Prepare annotations from mzTab-M

Description

Extracts structural annotations from mzTab-M tables and standardizes them for TIMA weighting and filtering steps.

Usage

```
prepare_annotations_mztab(  
  input = get_params(step = "prepare_annotations_mztab")$files$mztab$raw,  
  output = get_params(step =  
    "prepare_annotations_mztab")$files$annotations$prepared$structural$mztab,  
  str_stereo = get_params(step =  
    "prepare_annotations_mztab")$files$libraries$sop$merged$structures$stereo,  
  str_met = get_params(step =  
    "prepare_annotations_mztab")$files$libraries$sop$merged$structures$metadata,  
  str_tax_cla = get_params(step =  
    "prepare_annotations_mztab")$files$libraries$sop$merged$structures$taxonomies$cla,  
  str_tax_npc = get_params(step =  
    "prepare_annotations_mztab")$files$libraries$sop$merged$structures$taxonomies$npc,  
  strict = FALSE  
)
```

Arguments

input	character(1) Path to an mzTab-M file (.mztab or .json).
output	character(1) Output path for the prepared annotation table.
str_stereo	character(1) Path to the structure stereo lookup table.
str_met	character(1) Path to the structure metadata lookup table.
str_tax_cla	character(1) Path to the ClassyFire taxonomy table.
str_tax_npc	character(1) Path to the NPClassifier taxonomy table.
strict	logical(1) If TRUE, apply strict SME required-column validation during parsing.

Details

Annotation source priority:

1. SME (evidence) rows — highest specificity; mapped back to SML IDs via the SME_ID_REFS column when an SML section is present.
2. SML (small molecule summary) rows — used when no SME section exists.
3. SMF (feature) rows — fallback for feature-only files.

When no input is provided (or the file does not exist) an empty annotation table is written and the function returns silently.

Value

Character path to the prepared annotation file (invisibly when the empty-annotation fallback is used).

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:
# Prepare annotations from an mzTab-M file
prepare_annotations_mztab(
  input = "annotations.mztab",
  output = "prepared_annotations.tsv"
)

## End(Not run)
```

prepare_annotations_sirius

Prepare annotations SIRIUS

Description

Prepares SIRIUS annotation results (structure predictions, CANOPUS chemical classifications, and formula predictions) by harmonizing formats across SIRIUS versions (v5/v6), standardizing column names, and integrating with structure metadata.

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_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$npc,
max_analog_abs_mz_error = get_params(step =
  "prepare_annotations_sirius")$tools$sirius$max_analog_abs_mz_error
)

```

Arguments

input_directory [character](#) Character path to directory or zip file containing SIRIUS results.

output_ann [character](#) Character path for prepared structure annotation output.

output_can [character](#) Character path for prepared CANOPUS output.

output_for [character](#) Character path for prepared formula output.

sirius_version [character](#) Character SIRIUS version ("5" or "6").

str_stereo [character](#) Character path to structure stereochemistry file.

str_met [character](#) Character path to structure metadata file.

str_tax_cla [character](#) Character path to ClassyFire taxonomy file.

str_tax_npc [character](#) Character path to NPClassifier taxonomy file.

max_analog_abs_mz_error [numeric](#) Maximum allowed absolute m/z deviation (Da) for keeping SIRIUS spectral analog hits.

Details

This function:

- Validates inputs (version, paths, file existence).
- Loads SIRIUS output files (CANOPUS, formulas, structures, denovo, spectral matches).
- Harmonizes column names across SIRIUS v5 and v6.
- Joins with structure metadata (stereochemistry, names, taxonomy).
- Splits results into three output files: annotations, CANOPUS, formulas.
- Exports parameters and results.

If the input directory does not exist, returns an empty template with expected columns to ensure downstream compatibility.

Value

Character path to the prepared SIRIUS annotations file (invisible).

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:
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 MS2 spectral library matching results by standardizing column names, integrating structure metadata, and formatting for downstream TIMA annotation workflows. Handles various spectral matching result formats.

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_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	character Character string path to spectral matching results file
output	character Character string path for prepared spectral annotations output
str_stereo	character Character string path to structures stereochemistry file
str_met	character Character string path to structures metadata file
str_tax_cla	character Character string path to ClassyFire taxonomy file
str_tax_npc	character Character string path to NPClassifier taxonomy file

Value

Character string path to prepared spectral annotations

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big\(\)](#), [prepare_libraries_sop_big\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:
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_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 molecular network component (cluster) assignments by loading, standardizing, and formatting component IDs for each feature. Components represent groups of related features in the molecular network.

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	character Character vector of paths to input component files. Can be a single file or multiple files that will be combined.
output	character Character string path where prepared components should be saved

Value

Character string path to the prepared features' components file

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:  
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 molecular network edges by combining MS1-based and spectral similarity edges, adding entropy information, and standardizing column names. Edges represent relationships between features in the molecular network.

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

<code>input</code>	list Named list containing paths to edge files. Must have "ms1" and "spectral" elements pointing to respective edge files.
<code>output</code>	character Character string path where prepared edges should be saved
<code>name_source</code>	character Character string name of the source feature column in input files
<code>name_target</code>	character Character string name of the target feature column in input files

Value

Character string path to the prepared edges file

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_bigg\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:  
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

Prepares LC-MS feature tables by standardizing column names, filtering to top-intensity samples per feature, and formatting for downstream analysis. Supports multiple formats (mzmine, SLAW, SIRIUS).

Usage

```

prepare_features_tables(
  features = get_params(step = "prepare_features_tables")$files$features$raw,
  output = get_params(step = "prepare_features_tables")$files$features$prepared,
  candidates = get_params(step = "prepare_features_tables")$annotations$canidates$samples,
  name_adduct = get_params(step = "prepare_features_tables")$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	character Path to raw features file (CSV/TSV).
output	character Path where prepared features should be saved.
candidates	integer Number of top-intensity samples to retain per feature (default: from params; recommended <=5 to balance data size and coverage).
name_adduct	character Name of the adduct column in input.

name_features **character** Name of the feature ID column in input.
 name_rt **character** Name of the retention time column in input.
 name_mz **character** Name of the m/z column in input.

Value

character(1) Path to the prepared feature table (invisibly).

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_bigg\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:
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 time libraries by combining experimental and in silico predicted retention times from multiple sources (MGF files, CSV files). It standardizes retention time units, validates structures, and creates both RT libraries and pseudo structure-organism pairs for RT-based annotation.

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_na = get_params(step = "prepare_libraries_rt")$names$mgf$name,
col_rt = get_params(step = "prepare_libraries_rt")$names$mgf$retention_time,
col_sm = get_params(step = "prepare_libraries_rt")$names$mgf$smiles,
name_inchikey = get_params(step = "prepare_libraries_rt")$names$inchikey,
name_name = get_params(step = "prepare_libraries_rt")$names$compound_name,
name_rt = get_params(step = "prepare_libraries_rt")$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	character Character vector of paths to MGF files with experimental RT
mgf_is	character Character vector of paths to MGF files with in silico predicted RT
temp_exp	character Character vector of paths to CSV files with experimental RT
temp_is	character Character vector of paths to CSV files with in silico predicted RT
output_rt	character Character string path for prepared RT library output
output_sop	character Character string path for pseudo SOP output
col_ik	character Character string name of InChIKey column in MGF
col_na	character Character string name of compound name column in MGF
col_rt	character Character string name of retention time column in MGF
col_sm	character Character string name of SMILES column in MGF
name_inchikey	character Character string name of InChIKey column in CSV
name_name	character Character string name of compound name column in CSV
name_rt	character Character string name of retention time column in CSV
name_smiles	character Character string name of SMILES column in CSV
unit_rt	character Character string RT unit: "seconds" or "minutes"

Value

Character string path to the prepared retention time library

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_sop_bigg\(\)](#), [prepare_libraries_sop_closed\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

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

## End(Not run)
```

```
prepare_libraries_sop_bigg
```

Prepare libraries of structure organism pairs BiGG

Description

This function prepares BiGG (Biochemical, Genetic and Genomic) structure-organism pairs by querying BiGG models and PubChem for metabolite information, extracting chemical structures, and formatting for TIMA annotation workflows.

****Biota organism**:** This function creates a special "Biota" organism for metabolites present in all models (shared core metabolism). These structures represent universal biochemical pathways found across all life forms and are always assigned maximum biological score during annotation, regardless of sample taxonomy. The Biota organism has `organism_taxonomy_01domain = "Biota"` and `ottid = 0`.

Usage

```
prepare_libraries_sop_bigg(
  bigg_doi = "10.1093/nar/gkv1049",
  bigg_models = list(`Escherichia coli` = c(model_id = "iML1515", doi =
    "10.1038/nbt.3956"), `Saccharomyces cerevisiae` = c(model_id = "iMM904", doi =
    "10.1186/1752-0509-3-37"), `Homo sapiens` = c(model_id = "Recon3D", doi =
    "10.1038/nbt.4072")),
  bigg_url = "https://bigg.ucsd.edu/static/models/",
  output = get_params(step =
    "prepare_libraries_sop_bigg")$files$libraries$sop$prepared$bigg
)
```

Arguments

<code>bigg_doi</code>	character Character string DOI for BiGG database reference
<code>bigg_models</code>	list Named list of BiGG models with organism names as keys and named character vectors containing "model_id" and "doi" as values
<code>bigg_url</code>	character Character string base URL for BiGG models API
<code>output</code>	character Character string path for prepared BiGG library output

Value

Character string path to prepared BiGG structure-organism pairs

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_closed\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

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

## End(Not run)
```

prepare_libraries_sop_closed

Prepare libraries of structure organism pairs CLOSED

Description

This function prepares closed (private/restricted) structure- organism pair libraries by formatting columns, rounding values, and standardizing structure. Falls back to an empty template if the closed resource is not accessible.

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	character Character string path to input closed library file
output	character Character string path where prepared library should be saved

Value

Character string path to the prepared structure-organism pairs library

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:
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

This function prepares ECMDB (E. coli Metabolome Database) structure-organism pairs by parsing JSON data, extracting metabolite information, and formatting for TIMA workflows. Handles E. coli metabolite data with structures.

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	character Character string path to ECMDB JSON zip file
output	character Character string path for prepared ECMDB library output

Value

Character string path to prepared ECMDB structure-organism pairs

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big](#), [prepare_libraries_sop_closed\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:
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 HMDB (Human Metabolome Database) structure-organism pairs by parsing SDF files, extracting metadata, and formatting for TIMA annotation workflows.

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	character Character string path to HMDB SDF zip file
output	character Character string path for prepared HMDB library output

Value

Character string path to prepared HMDB structure-organism pairs

See Also

Other preparation: `prepare_annotations_gnps()`, `prepare_annotations_mzmine()`, `prepare_annotations_mztab()`, `prepare_annotations_sirius()`, `prepare_annotations_spectra()`, `prepare_features_components()`, `prepare_features_edges()`, `prepare_features_tables()`, `prepare_libraries_rt()`, `prepare_libraries_sop_big`, `prepare_libraries_sop_closed()`, `prepare_libraries_sop_ecmdb()`, `prepare_libraries_sop_lotus()`, `prepare_libraries_sop_merged()`, `prepare_libraries_sop_pubchemlite()`, `prepare_libraries_spectra()`, `prepare_params()`, `prepare_taxa()`, `read_mztab()`

Examples

```
## Not run:
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. It standardizes columns, extracts 2D InChIKeys, rounds numeric values, and removes duplicates.

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

<code>input</code>	character Character string path to the raw LOTUS data file
<code>output</code>	character Character string path for the prepared output file

Value

Character string path to the prepared structure-organism pairs library file

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big](#), [prepare_libraries_sop_closed\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:
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 merges all structure-organism pair libraries (LOTUS, HMDB, ECMDB, etc.) into a single comprehensive library. Can optionally filter by taxonomic level to create biologically-focused subsets. Also splits structures into separate metadata tables.

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$value,
  cache = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$structures$processed,
  npc_cache = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$structures$taxonomies$n,
  cla_cache = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$structures$taxonomies$c,
  output_key = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$keys,
  output_org_tax_ott = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$organisms$taxonomies$ott,
  output_str_can = get_params(step =
    "prepare_libraries_sop_merged")$files$libraries$sop$merged$structures$canonical,
```

```

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_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	character Character vector or list of paths to prepared library files
filter	logical Logical whether to filter the merged library by taxonomy
level	character Character string taxonomic rank for filtering (kingdom, phylum, family, genus, etc.)
value	character Character string taxon name(s) to keep (can use for multiple, e.g., 'Gentianaceae Apocynaceae')
cache	character Character string path to cache directory for processed SMILES
npc_cache	character Optional path to an additional NPClassifier taxonomy cache file (TSV/TSV.gz). Structures present in the merged library but missing NPClassifier taxonomy will be looked up in this cache. Expected columns: structure_smiles, structure_tax_npc_01pat, structure_tax_npc_02sup, structure_tax_npc_03cla. Alternative column names from external tools (e.g., pathway, superclass, class) are also supported.
cla_cache	character Optional path to an additional ClassyFire taxonomy cache file (TSV/TSV.gz). Structures present in the merged library but missing ClassyFire taxonomy will be looked up in this cache. Expected columns: structure_inchikey, structure_tax_cla_chemontid, structure_tax_cla_01kin, structure_tax_cla_02sup, structure_tax_cla_03cla, structure_tax_cla_04dirpar. Alternative column names (e.g., inchikey, chemontid, kingdom, superclass, class, directparent) are also supported.
output_key	character Character string path for output keys file
output_org_tax_ott	character Character string path for organisms taxonomy (OTT) file
output_str_can	character Character string path for structures canonical SMILES file
output_str_stereo	character Character string path for structures stereochemistry file
output_str_met	character Character string path for structures metadata file
output_str_tax_cla	character Character string path for ClassyFire taxonomy file
output_str_tax_npc	character Character string path for NPClassifier taxonomy file

Details

Creates merged library by combining all available SOP sources, optionally filtering by taxonomic criteria (e.g., only Gentianaceae). Splits output into structures metadata, names, taxonomy, and organisms.

Value

Character string path to the prepared merged SOP library

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big](#), [prepare_libraries_sop_closed\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:
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_sop_pubchemlite

Prepare libraries of structure organism pairs PubChem Lite

Description

This function prepares the PubChem Lite CCSbase export for exposomics as a xenobiotic structure-organism pairs library.

Usage

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

Arguments

input **character** Character string path to PubChem Lite CSV file
output **character** Character string path for prepared SOP output

Value

Character string path to prepared SOP file

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big](#), [prepare_libraries_sop_closed\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

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

prepare_libraries_spectra

Prepare libraries of spectra

Description

Prepares spectral libraries for matching by importing, harmonizing, and splitting spectra by polarity. Exports results as Spectra RDS files (pos/neg) and a structure-organism pair (SOP) table.

Usage

```
prepare_libraries_spectra(
  input = get_params(step = "prepare_libraries_spectra")$files$libraries$spectral$raw,
  min_fragments = get_params(step =
    "prepare_libraries_spectra")$ms$thresholds$ms2$min_fragments,
  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$compound_id,
  col_in = get_params(step = "prepare_libraries_spectra")$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$inchikey_connectivity_layer,
  col_na = get_params(step = "prepare_libraries_spectra")$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
)
```

Arguments

input	character Character vector of file paths containing spectral data.
min_fragments	integer Minimum number of fragment peaks a spectrum must have after cleaning to be retained (default: 2).
nam_lib	character Character library name for metadata.
col_ad	character Name of the adduct column in MGF.
col_ce	character Name of the collision energy column in MGF.
col_ci	character Name of the compound ID column in MGF.
col_in	character Name of the InChI column in MGF.
col_io	character Name of the InChI without stereo column in MGF.
col_ik	character Name of the InChIKey column in MGF.
col_il	character Name of the InChIKey connectivity layer column in MGF.
col_na	character Name of the name column in MGF.
col_po	character Name of the polarity column in MGF.
col_sm	character Name of the SMILES column in MGF.
col_sn	character Name of the SMILES without stereo column in MGF.
col_si	character Name of the spectrum ID column in MGF.
col_sp	character Name of the SPLASH column in MGF.
col_sy	character Name of the synonyms column in MGF.

Details

This function:

- Checks if output files already exist (idempotent).
- Imports spectral data from input files.
- Extracts and harmonizes spectra for positive and negative modes.
- Fixes precursor m/z and InChIKey connectivity layer issues.
- Exports polarity-specific Spectra objects and SOP table.
- Returns empty templates if input files are missing.

Value

Character vector with paths to prepared library files (invisible).

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big](#), [prepare_libraries_sop_closed\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

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

## End(Not run)
```

```
prepare_params
```

```
Prepare workflow parameters
```

Description

Prepares and validates main parameters for the TIMA workflow. Loads YAML configuration files, extracts all parameters, and sets up the parameter structure for downstream analysis steps.

Usage

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

Arguments

`params_small` [list](#) List of basic parameters for the workflow
`params_advanced` [list](#) List of advanced parameters for the workflow
`step` [character](#) Workflow step identifier (default: NA)

Value

Character vector of paths to YAML files containing prepared parameters

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big](#), [prepare_libraries_sop_closed\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_taxa\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:  
# Prepare parameters for TIMA workflow  
param_files <- prepare_params(  
  params_small = get_params(step = "prepare_params"),  
  params_advanced = get_params(step = "prepare_params_advanced")  
)  
  
# Parameters are exported to timestamped files  
# and can be loaded later for reproducibility  
  
## End(Not run)
```

prepare_taxa

Prepare taxa

Description

This function prepares taxonomic information for features by matching organism names to Open Tree of Life taxonomy. Can attribute all features to a single organism or distribute them across multiple organisms based on relative intensities in samples.

Usage

```
prepare_taxa(
  input = get_params(step = "prepare_taxa")$files$features$prepared,
  extension = get_params(step = "prepare_taxa")$names$extension,
  name_filename = get_params(step = "prepare_taxa")$names$filename,
  colname = get_params(step = "prepare_taxa")$names$taxon,
  metadata = get_params(step = "prepare_taxa")$files$metadata$raw,
  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	character Character string path to features file with intensities
extension	logical Logical whether column names contain file extensions
name_filename	character Character string name of filename column in metadata
colname	character Character string name of column with biological source info
metadata	character Character string path to metadata file with organism info
org_tax_ott	character Character string path to Open Tree of Life taxonomy file
output	character Character string path for output file
taxon	character Character string organism name to enforce for all features (e.g., "Homo sapiens"). If provided, overrides metadata-based assignment.

Details

Depending on whether features are aligned between samples from various organisms, this function either: - Attributes all features to a single organism (if taxon specified), or - Attributes features to multiple organisms based on their relative intensities across samples (using metadata)

Value

Character string path to the prepared taxa file

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big\(\)](#), [prepare_libraries_sop_closed\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [read_mztab\(\)](#)

Examples

```
## Not run:
copy_backbone()
go_to_cache()
github <- "https://raw.githubusercontent.com/"
repo <- "taxonomicallyinformedannotation/tima-example-files/main/"
dir <- paste0(github, repo)
org_tax_ott <- paste0(
  "data/interim/libraries/",
  "sop/merged/organisms/taxonomies/ott.tsv"
)
get_file(url = paste0(dir, org_tax_ott), export = org_tax_ott)
get_file(
  url = paste0(dir, "data/interim/features/example_features.tsv"),
  export = get_params(step = "prepare_taxa")$files$features$prepared
)
prepare_taxa(
  taxon = "Homo sapiens",
  org_tax_ott = org_tax_ott
)
unlink("data", recursive = TRUE)

## End(Not run)
```

process_smiles

Process SMILES strings

Description

Processes SMILES using RDKit (via Python) to standardize structures, generate InChIKeys, calculate molecular properties, and extract 2D representations. Results are cached to avoid reprocessing.

Usage

```
process_smiles(df, smiles_colname = "structure_smiles_initial", cache = NULL)
```

Arguments

df [data.frame](#) Data frame containing SMILES strings

smiles_colname [character](#) Column name containing SMILES (default: "structure_smiles_initial")

cache [character](#) Path to cached processed SMILES file, or NULL to skip caching

Value

Data frame with processed SMILES including InChIKey, molecular formula (with isotopes shown), exact mass (with isotope contributions), 2D SMILES, xLogP, and connectivity layer

Examples

```
## Not run:
# Natural compound
df <- data.frame(
  structure_smiles_initial = "OC[C@H]1OC(O)[C@H](O)[C@H](O)[C@H]1O"
)
result <- process_smiles(df)
# Formula: C6H12O6, Mass: 180.063 Da

# Isotope-labeled compound (4x 13C)
df_labeled <- data.frame(
  structure_smiles_initial = "OC[13C@H]1OC(O)[13C@H](O)[13C@H](O)[13C@H]1O"
)
result_labeled <- process_smiles(df_labeled)
# Formula: C2[13C]4H12O6 (isotopes shown separately)
# Mass: 184.077 Da (difference of ~4.013 Da from natural)
# SMILES preserves [13C] notation
# InChIKey differs from natural glucose

## End(Not run)
```

read_mztab

Read mzTab-M and export TIMA-compatible files

Description

Parses mzTab-M plain-text files (v2.0.0–M) and exports TIMA-ready feature, optional spectra (MGF), and optional metadata tables.

Usage

```
read_mztab(
  input,
  output_features,
  output_spectra = NULL,
  output_metadata = NULL,
  name_features = "feature_id",
  name_rt = "rt",
  name_mz = "mz",
  name_adduct = "adduct",
  strict = FALSE
)
```

Arguments

input	character(1) Path to an mzTab-M file (.mztab or .json for rmzTabM / progenesis / MetaboScape JSON layouts).
-------	---

output_features	character(1) Output path for the feature table (.tsv or .csv).
output_spectra	character(1) NULL Output path for the MGF spectra file. NULL (default) skips spectral export.
output_metadata	character(1) NULL Output path for the metadata table. NULL (default) skips metadata export.
name_features	character(1) Name for the feature identifier column in the output feature table.
name_rt	character(1) Name for the retention time column (minutes).
name_mz	character(1) Name for the precursor m/z column.
name_adduct	character(1) Name for the adduct column.
strict	logical(1) If TRUE, enforce stricter SME required-column checks during validation.

Details

Two spectrum export modes are supported:

Embedded spectra (masster COM MGF) When the mzTab-M file contains masster-style COM\tMGH / COM\tMGF lines, real MS2 spectra are extracted. Each entry carries a FEATURE_ID= field so that [get_spectra_ids\(\)](#) can map edges back to feature IDs.

Proxy MGF When no embedded spectra are found, a proxy MGF is generated with one dummy entry per feature (using the precursor m/z as the sole peak). Each entry carries both TITLE= and FEATURE_ID= set to the feature identifier.

Value

Named list with paths: \$features (always set), \$spectra and \$metadata (NULL when the corresponding output argument is NULL or the export step is skipped).

See Also

Other preparation: [prepare_annotations_gnps\(\)](#), [prepare_annotations_mzmine\(\)](#), [prepare_annotations_mztab\(\)](#), [prepare_annotations_sirius\(\)](#), [prepare_annotations_spectra\(\)](#), [prepare_features_components\(\)](#), [prepare_features_edges\(\)](#), [prepare_features_tables\(\)](#), [prepare_libraries_rt\(\)](#), [prepare_libraries_sop_big](#), [prepare_libraries_sop_closed\(\)](#), [prepare_libraries_sop_ecmdb\(\)](#), [prepare_libraries_sop_hmdb\(\)](#), [prepare_libraries_sop_lotus\(\)](#), [prepare_libraries_sop_merged\(\)](#), [prepare_libraries_sop_pubchemlite\(\)](#), [prepare_libraries_spectra\(\)](#), [prepare_params\(\)](#), [prepare_taxa\(\)](#)

Examples

```
## Not run:
# Read mzTab-M file and export TIMA-compatible tables
result <- read_mztab(
  input = "annotations.mztab",
  output_features = "features.tsv",
  output_spectra = "spectra.mgf",
  output_metadata = "metadata.tsv"
```

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

run_app

Run TIMA Shiny app

Description

Launches the TIMA Shiny web application for interactive metabolite annotation. Automatically detects Docker containers and adjusts network settings accordingly.

Usage

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

Arguments

host	character Host/IP address to listen on. Default: "127.0.0.1" (localhost). Use "0.0.0.0" to allow external connections.
port	integer Port number to listen on. Default: 3838. Valid range: 1-65535.
browser	logical Whether to automatically launch a web browser when starting the app. Default: TRUE. Automatically set to FALSE in Docker.
reinstall	logical Whether to automatically reinstall TIMA. Default: TRUE.

Value

NULL (invisibly). Launches the Shiny app as a side effect.

See Also

Other workflow: [change_params_small\(\)](#), [create_components\(\)](#), [create_edges\(\)](#), [create_edges_spectra\(\)](#), [go_to_cache\(\)](#), [install\(\)](#), [install_tima\(\)](#), [run_tima\(\)](#), [tima_full\(\)](#), [validate_inputs\(\)](#)

Examples

```
## Not run:
# Launch app on localhost
run_app()

# Launch on custom port
run_app(port = 8080)

# Allow external connections (useful in Docker)
run_app(host = "0.0.0.0", port = 3838)

## End(Not run)
```

run_tima*Run Complete TIMA Workflow*

Description

Executes the full Taxonomically Informed Metabolite Annotation (TIMA) workflow from start to finish. This includes data preparation, library loading, annotation, weighting, and output generation. The function runs the targets pipeline and archives logs with timestamps for reproducibility.

Usage

```
run_tima(  
  target_pattern = "^(ann_wei|exp_mzt)$",  
  log_file = "tima.log",  
  clean_old_logs = TRUE,  
  log_level = "info"  
)
```

Arguments

target_pattern [character](#) Regex pattern for target selection. Default: "^(ann_weilexp_mzt)\$" (annotation preparation + mzTab export)

log_file [character](#) Path to log file. Default: "tima.log"

clean_old_logs [logical](#) Remove old log file before starting. Default: TRUE

log_level [character](#) or [numeric](#) Logging verbosity level. Can be one of: "trace", "debug", "info", "warn", "error", "fatal" or numeric values: TRACE=600, DEBUG=500, INFO=400, WARN=300, ERROR=200, FATAL=100. Default: "info" (400). Use "debug" for detailed troubleshooting.

Details

The workflow performs the following steps:

- Initializes logging and timing
- Navigates to cache directory
- Executes the targets pipeline (annotation preparation + mzTab export)
- Archives timestamped logs to data/processed/

Value

Invisible NULL. Executes workflow as side effect and creates timestamped log files in data/processed/

See Also

Other workflow: [change_params_small\(\)](#), [create_components\(\)](#), [create_edges\(\)](#), [create_edges_spectra\(\)](#), [go_to_cache\(\)](#), [install\(\)](#), [install_tima\(\)](#), [run_app\(\)](#), [tima_full\(\)](#), [validate_inputs\(\)](#)

Examples

```
## Not run:
# Run full workflow with defaults (INFO level)
run_tima()

# Run with debug logging for troubleshooting
run_tima(log_level = "debug")

# Run with minimal logging (warnings and errors only)
run_tima(log_level = "warn")

# Run with custom target pattern
run_tima(target_pattern = "^prepare_")

# Preserve existing logs
run_tima(clean_old_logs = FALSE)

# Combine multiple options
run_tima(
  target_pattern = "^ann_",
  log_level = "debug",
  clean_old_logs = FALSE
)

## End(Not run)
```

tima_full

Run Complete TIMA Workflow (DEPRECATED)

Description

DEPRECATED: This function has been renamed to `run_tima`. Please use `run_tima()` instead. `tima_full()` will be removed in a future version.

Usage

```
tima_full(
  target_pattern = "^(ann_wei|exp_mzt)$",
  log_file = "tima.log",
  clean_old_logs = TRUE,
  log_level = "info"
)
```

Arguments

target_pattern	Character. Regex pattern for target selection. Default: "^(ann_weilexp_mzt)\$"
log_file	Character. Path to log file. Default: "tima.log"
clean_old_logs	Logical. Remove old log file before starting. Default: TRUE
log_level	Character or numeric. Logging verbosity level. Default: "info"

Details

This function is deprecated as of TIMA version 2.12.0 (November 2025). It now simply calls [run_tima](#) with all arguments passed through, but issues a deprecation warning.

Migration Guide:

- Old: `tima_full(target_pattern = "^(ann_wei|exp_mzt)$")`
- New: `run_tima(target_pattern = "^(ann_wei|exp_mzt)$")`

All parameters and behavior are identical between the two functions.

Value

Invisible NULL (same as [run_tima](#))

See Also

[run_tima](#) for the current function

Other workflow: [change_params_small\(\)](#), [create_components\(\)](#), [create_edges\(\)](#), [create_edges_spectra\(\)](#), [go_to_cache\(\)](#), [install\(\)](#), [install_tima\(\)](#), [run_app\(\)](#), [run_tima\(\)](#), [validate_inputs\(\)](#)

Examples

```
## Not run:
# DEPRECATED - Use run_tima() instead
# tima_full()

# RECOMMENDED
run_tima()

## End(Not run)
```

transform_score_sirius_csi
Transform SIRIUS CSI score

Description

This function transforms SIRIUS CSI (Compound Structure Identification) scores using a sigmoid function. The transformation maps raw scores to a 0-1 range for better interpretability.

Usage

```
transform_score_sirius_csi(csi_score = NULL, K = 100, scale = 20)
```

Arguments

csi_score	numeric Numeric SIRIUS CSI score (expected mostly ≤ 0 ; can be negative, NA, NULL, or absent)
K	numeric Numeric shift parameter to adjust the sigmoid center (default: 100, midpoint at score = -100)
scale	numeric Numeric scale parameter controlling sigmoid steepness (default: 20)

Details

This is an experimental transformation not officially approved by SIRIUS developers. The sigmoid function is: $1 / (1 + \exp(-(score + K) / scale))$

SIRIUS CSI:FingerID scores are expected to be log-likelihood-like values on $(-\infty, 0]$, where values closer to 0 are better. A practical rule of thumb is:

```
\itemize{
  \item score > -10: excellent/awesome
  \item score > -100: acceptable/okay
  \item score <= -100: weak/low confidence
}
```

The defaults $K = 100$ and $scale = 20$ place the sigmoid midpoint at score = -100 and strongly reward scores near 0:

```
\itemize{
  \item score = -10 -> ~0.989
  \item score = -100 -> 0.500
  \item score = -200 -> ~0.007
}
```

Previous defaults ($K = 50$, $scale = 10$) placed the midpoint at -50 and compressed the useful range to $[-70, -30]$, mapping most realistic SIRIUS hits to near-zero scores.

Value

Numeric transformed score in the range (0, 1), or NA if input is NA/NULL/absent

Examples

```
## Not run:
# Transform a single score
transform_score_sirius_csi(csi_score = -100)

# Transform with custom parameters
transform_score_sirius_csi(csi_score = -100, K = 100, scale = 20)

# Transformation
scores <- c(-300, -100, -10, -1, 0)
transform_score_sirius_csi(csi_score = scores)
```

```
# Handle NA values
scores_with_na <- c(-100, NA, -10, -300)
transform_score_sirius_csi(csi_score = scores_with_na)

# Handle missing/absent score
transform_score_sirius_csi()

## End(Not run)
```

validate_inputs	<i>Validate Input Data</i>
-----------------	----------------------------

Description

Standalone command to validate all input data before starting the TIMA pipeline. This helps catch issues early and avoid wasting time on library downloads and processing.

Usage

```
validate_inputs(
  features = NULL,
  spectra = NULL,
  metadata = NULL,
  sirius = NULL,
  filename_col = "filename",
  organism_col = "organism",
  feature_col = "feature_id"
)
```

Arguments

features	character Character path to features CSV/TSV file
spectra	character Character path to MGF spectra file
metadata	character Character path to metadata file
sirius	character Character path to SIRIUS output directory or ZIP file
filename_col	character Character name of filename column (default: "filename")
organism_col	character Character name of organism column (default: "organism")
feature_col	character Character name of feature ID column (default: "feature_id")

Value

Invisible TRUE if all checks pass, stops with error otherwise

See Also

Other workflow: [change_params_small\(\)](#), [create_components\(\)](#), [create_edges\(\)](#), [create_edges_spectra\(\)](#), [go_to_cache\(\)](#), [install\(\)](#), [install_tima\(\)](#), [run_app\(\)](#), [run_tima\(\)](#), [tima_full\(\)](#)

Examples

```
## Not run:
# Validate all inputs before starting pipeline
validate_inputs(
  features = "data/features.csv",
  spectra = "data/spectra.mgf",
  sirius = "data/sirius_output"
)

# Validate with metadata consistency check
validate_inputs(
  features = "data/features.csv",
  metadata = "data/metadata.tsv"
)

## End(Not run)
```

weight_annotations	<i>Weight annotations</i>
--------------------	---------------------------

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$processed,
  candidates_neighbors = get_params(step =
    "weight_annotations")$annotations$candidates$neighbors,
  candidates_final = get_params(step = "weight_annotations")$annotations$candidates$final,
  best_percentile = get_params(step =
    "weight_annotations")$annotations$candidates$best_percentile,
  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$variety,
score_biological_biota = get_params(step =
  "weight_annotations")$weights$biological$biota,
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_evidence = get_params(step = "weight_annotations")$options$high_evidence,
remove_ties = get_params(step = "weight_annotations")$options$remove_ties,
summarize = get_params(step = "weight_annotations")$options$summarize,
pattern = get_params(step = "weight_annotations")$files$pattern,
force = get_params(step = "weight_annotations")$options$force,
xrefs_file = NULL
)

```

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_neighbors	Number of neighbors candidates to keep
candidates_final	Number of final candidates to keep
best_percentile	Numeric percentile threshold (0-1) for selecting top candidates within each feature (default: 0.9). Used for consistent filtering between mini and filtered outputs.
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_biological_biota
Score for a Biota match (should be the highest, special)

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_evidence	Report high evidence candidates only. BOOLEAN
remove_ties	Remove ties. BOOLEAN
summarize	Summarize results (1 row per feature). BOOLEAN
pattern	Pattern to identify your job. STRING
force	Force parameters. Use it at your own risk
xrefs_file	Optional character path to xrefs file from get_compounds_xrefs() . If provided, external database identifiers will be added to results.

Value

The path to the weighted annotations

See Also

[annotate_masses](#) [weight_bio](#) [weight_chemo](#)

Other annotation: [annotate_masses\(\)](#), [annotate_spectra\(\)](#), [filter_annotations\(\)](#), [write_mztab\(\)](#)

Examples

```
## Not run:
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 <- paste0(
  "data/interim/libraries/",
```

```

    "sop/merged/organisms/taxonomies/ott.tsv"
  )
  str_stereo <- paste0(
    "data/interim/libraries/",
    "sop/merged/structures/stereo.tsv"
  )
  annotations <- paste0(
    "data/interim/annotations/",
    "example_annotationsFiltered.tsv"
  )
  canopus <- paste0(
    "data/interim/annotations/",
    "example_canopusPrepared.tsv"
  )
  formula <- paste0(
    "data/interim/annotations/",
    "example_formulaPrepared.tsv"
  )
  components <- paste0(
    "data/interim/features/",
    "example_componentsPrepared.tsv"
  )
  edges <- paste0(
    "data/interim/features/",
    "example_edges.tsv"
  )
  taxa <- paste0(
    "data/interim/taxa/",
    "example_taxed.tsv"
  )
  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)

```

write_mztab

*Write TIMA results as mzTab-M***Description**

Exports TIMA weighted-annotation results to mzTab-M 2.1.0 plain-text format. The output is a compliant mzTab-M file containing:

- **MTD** – metadata (software, database, instrument, evidence measures, ms_run, sample, assay, study_variable).
- **SMF** – one row per chromatographic feature (feature_id, m/z, RT).
- **SME** – one row per identification evidence (candidate annotation).
- **SML** – one row per unique compound, linking all associated SMF and SME rows.

Usage

```
write_mztab(
  input = get_params(step = "write_mztab")$files$annotations$processed,
  output = get_params(step = "write_mztab")$files$output$mztab,
  ms_run_location = "null",
  ms_run_format = "null",
  ms_run_id_format = "null",
  polarity = NULL,
  instrument = NULL,
  sample_name = NULL,
  publication = NULL,
  title = "TIMA annotation results",
  description = paste0("Annotation results produced by Taxonomically Informed ",
    "Metabolomics Annotation (TIMA)."),
  software_version = as.character(utils::packageVersion("tima")),
  contact = NULL,
  xrefs_file = NULL,
  edges_file = NULL,
  base_mztab = NULL
)
```

Arguments

input	character Path to TIMA results file produced by weight_annotations().
output	character Destination path for the mzTab-M file (.mztab extension recommended).
ms_run_location	character URI/path to the originating raw data file (used in MTD ms_run[1]-location). Defaults to "null".
ms_run_format	character CV Param string for the raw file format, e.g. "[MS, MS:1000584, mzML file,]". Defaults to "null" when not known.

ms_run_id_format	character CV Param string for the spectrum native-ID format, e.g. "[MS, MS:1000776, scan number only nativeID format,]". Defaults to "null".
polarity	character NULL Scan polarity of the MS run. One of "positive", "negative", or NULL/"null" (unknown / data-dependent). Used to populate ms_run[1]-scan_polarity with the correct PSI-MS CV term (MS:1000130 positive, MS:1000129 negative).
instrument	character NULL CV Param string for the mass spectrometer model, e.g. "[MS, MS:1001742, LTQ Orbitrap Velos,]". When NULL the instrument[1] block is omitted.
sample_name	character NULL Free-text sample name written to sample[1]-description. When NULL, a generic metabolomics description is used.
publication	character NULL Bibliographic reference for the study (PubMed URL or DOI). When provided, emitted as MTD publication[1].
title	character Free-text study title written to MTD title.
description	character Free-text experiment description written to MTD description.
software_version	character Version string for the software entry. Defaults to the installed TIMA package version.
contact	list NULL Optional contact information list with fields name, email, and optionally affiliation. When supplied, the corresponding MTD contact[1]-* lines are emitted.
xrefs_file	character NULL Optional path to a cross-references TSV produced by <code>get_compounds_xrefs()</code> . When provided, the uri fields in SME/SML are enriched with external database URLs (Wikidata preferred), and additional MTD database[n] blocks are registered for each unique xref prefix found in the data.
edges_file	character NULL Optional path to an edge table (for example from data/interim). When provided, all edge rows are embedded in the mzTab text export as COM extension lines (TIMA edges) so graph information lives in the same artifact.
base_mztab	character NULL Optional existing mzTab-M file to complement. Existing SML/SMF/SME rows are preserved and new TIMA rows are appended without duplication. Non-managed lines/sections are passed through unchanged.

Details

The function intentionally writes in **Summary** mode because TIMA is an annotation/prioritization tool and does not guarantee complete quantification matrices. Fields that have no TIMA equivalent (e.g. full InChI, spectra_ref) are written as null.

TIMA columns are mapped to canonical mzTab fields where a direct equivalent exists; only truly unmapped columns fall back to opt_global_* to keep downstream consumers happy:

- **SME section** – candidate-level columns with no canonical field (e.g. SIRIUS subscores, similarity forward/reverse, m/z error) become opt_global_*. Feature-level columns are **not** repeated here; they belong to the SMF section.
- **SMF section** – all feature_* columns beyond feature_mz and feature_rt (spectrum entropy, spectrum peaks, predicted taxonomy class/NPC scores...) are exported as opt_global_* in the SMF row, making them available without polluting SME.

mzTab-M reliability mapping:

Reliability levels follow the Metabolomics Standards Initiative (MSI) scale:

- **1** – confirmed (score ≥ 0.7 with spectral library evidence)
- **2** – probable (score ≥ 0.5 ; or spectral match, any score)
- **3** – putative (score ≥ 0.2)
- **4** – unambiguous compound class only (everything else)

id_confidence_measure columns:

Four TIMA-specific evidence measures are exported as `id_confidence_measure[1..4]` in the MTD section and as additional columns in the SME section. All use the TIMA user-controlled CV namespace (no PSI-MS accession exists for these composite scores):

- [1] – `score_final` (combined TIMA score; TIMA:001)
- [2] – `score_biological` (taxonomic score; TIMA:002)
- [3] – `score_chemical` (chemical consistency; TIMA:003)
- [4] – `candidate_score_similarity` (spectral similarity; TIMA:004; omitted when no spectral evidence is present)

Ontology alignment:

- `ms_level` uses PSI-MS accessions MS:1000579 (MS1) and MS:1000580 (MS2).
- `scan_polarity` uses MS:1000130 (positive) and MS:1000129 (negative) when polarity is supplied.
- `retention_time_in_seconds` is declared in `colunit-small_molecule_feature` with UO accession UO:0000010.
- `theoretical_neutral_mass` is declared with UO:0000221 (dalton).
- `spectra_ref` is formatted as `ms_run[1]:{spectrum_native_id}`.
- `instrument[1]` is populated from the `instrument` parameter using a PSI-MS CV Param when provided.
- `quantification_method` is set to [MS, MS:1001834, LC-MS label-free quantitation analysis,] for untargeted metabolomics (per PSI-MS ontology).
- `assay[1]-quantification_reagent` is set to [MS, MS:1002038, unlabeled sample,] (no labelling used by TIMA).
- `sample[1]` defaults to a metabolite mixture Param; can be overridden via the `sample_name` parameter.
- `publication` emits a formatted citation when `publication` is supplied.
- The software entry includes the TIMA repository URL as a `software[1]-setting` for machine-readable provenance.

Value

Character path to the written mzTab-M file (invisibly).

See Also

Other annotation: [annotate_masses\(\)](#), [annotate_spectra\(\)](#), [filter_annotations\(\)](#), [weight_annotations\(\)](#)

Examples

```
## Not run:
write_mztab(
  input = "annotations.tsv",
  output = "annotations.mztab"
)

## End(Not run)
```

Index

* annotation

- annotate_masses, 6
- annotate_spectra, 9
- filter_annotations, 26
- weight_annotations, 76
- write_mztab, 82

* chemical-classification

- process_smiles, 67

* data-retrieval

- get_compounds_xrefs, 27
- get_example_files, 28
- get_file, 30
- get_gnps_tables, 31
- get_last_version_from_zenodo, 32
- get_organism_taxonomy_ott, 33

* mass-spectrometry

- calculate_mass_of_m, 12
- calculate_mz_from_mass, 14
- calculate_similarity, 15
- harmonize_adducts, 36
- import_spectra, 37
- parse_adduct, 40

* preparation

- prepare_annotations_gnps, 41
- prepare_annotations_mzmine, 42
- prepare_annotations_mztab, 43
- prepare_annotations_sirius, 45
- prepare_annotations_spectra, 47
- prepare_features_components, 48
- prepare_features_edges, 50
- prepare_features_tables, 51
- prepare_libraries_rt, 52
- prepare_libraries_sop_big, 54
- prepare_libraries_sop_closed, 55
- prepare_libraries_sop_ecmdb, 56
- prepare_libraries_sop_hmdb, 57
- prepare_libraries_sop_lotus, 58
- prepare_libraries_sop_merged, 59
- prepare_libraries_sop_pubchemlite,

61

- prepare_libraries_spectra, 62
- prepare_params, 64
- prepare_taxa, 65
- read_mztab, 68

* workflow

- change_params_small, 16
- create_components, 22
- create_edges, 23
- create_edges_spectra, 24
- go_to_cache, 34
- install, 38
- install_tima, 39
- run_app, 70
- run_tima, 71
- tima_full, 72
- validate_inputs, 75

adduct_to_string, 5

annotate_masses, 6

annotate_masses(), 11, 26, 80, 84

annotate_spectra, 9

annotate_spectra(), 9, 26, 80, 84

build_components_from_edges, 11

calculate_mass_of_m, 12

calculate_mass_of_m(), 14, 16, 36, 37, 40

calculate_mz_from_mass, 14

calculate_mz_from_mass(), 13, 16, 36, 37, 40

calculate_similarity, 15

calculate_similarity(), 13, 14, 36, 37, 40

change_params_small, 16

change_params_small(), 22, 24, 25, 35, 38, 39, 70, 71, 73, 75

character, 9, 13, 15, 25, 30, 34, 35, 38–41,

43, 46, 47, 49–58, 60, 62, 63, 65–67, 70, 71, 75

clean_chemo, 18

- copy_backbone, 21
- create_components, 22
- create_components(), 18, 24, 25, 35, 38, 39, 70, 71, 73, 75
- create_edges, 23
- create_edges(), 18, 22, 25, 35, 38, 39, 70, 71, 73, 75
- create_edges_spectra, 24
- create_edges_spectra(), 18, 22, 24, 35, 38, 39, 70, 71, 73, 75
- data.frame, 34, 67
- filter_annotations, 26
- filter_annotations(), 9, 11, 80, 84
- filter_high_evidence_only, 20
- get_compounds_xrefs, 27
- get_compounds_xrefs(), 29, 30, 32–34, 80, 83
- get_example_files, 28
- get_example_files(), 28, 30, 32–34
- get_example_sirius, 29
- get_file, 30
- get_file(), 28, 29, 32–34
- get_gnps_tables, 31
- get_gnps_tables(), 28–30, 33, 34
- get_last_version_from_zenodo, 32
- get_last_version_from_zenodo(), 28–30, 32, 34
- get_organism_taxonomy_ott, 33
- get_organism_taxonomy_ott(), 28–30, 32, 33
- get_spectra_ids(), 69
- go_to_cache, 34
- go_to_cache(), 18, 22, 24, 25, 38, 39, 70, 71, 73, 75
- harmonize_adducts, 36
- harmonize_adducts(), 13, 14, 16, 37, 40
- import_spectra, 37
- import_spectra(), 13, 14, 16, 36, 40
- install, 38
- install(), 18, 22, 24, 25, 35, 39, 70, 71, 73, 75
- install_tima, 39
- install_tima(), 18, 22, 24, 25, 35, 38, 70, 71, 73, 75
- integer, 10, 25, 30, 51, 63, 70
- list, 29, 50, 54, 65
- logical, 10, 15, 34, 38, 39, 60, 66, 70, 71
- matrix, 15
- numeric, 9, 10, 13, 15, 25, 46, 71, 74
- parse_adduct, 40
- parse_adduct(), 13, 14, 16, 36, 37
- prepare_annotations_gnps, 41
- prepare_annotations_gnps(), 43, 45, 46, 48–50, 52, 53, 55–59, 61, 62, 64–66, 69
- prepare_annotations_mzmine, 42
- prepare_annotations_mzmine(), 42, 45, 46, 48–50, 52, 53, 55–59, 61, 62, 64–66, 69
- prepare_annotations_mztab, 43
- prepare_annotations_mztab(), 42, 43, 46, 48–50, 52, 53, 55–59, 61, 62, 64–66, 69
- prepare_annotations_sirius, 45
- prepare_annotations_sirius(), 42, 43, 45, 48–50, 52, 53, 55–59, 61, 62, 64–66, 69
- prepare_annotations_spectra, 47
- prepare_annotations_spectra(), 42, 43, 45, 46, 49, 50, 52, 53, 55–59, 61, 62, 64–66, 69
- prepare_features_components, 48
- prepare_features_components(), 42, 43, 45, 46, 48, 50, 52, 53, 55–59, 61, 62, 64–66, 69
- prepare_features_edges, 50
- prepare_features_edges(), 42, 43, 45, 46, 48, 49, 52, 53, 55–59, 61, 62, 64–66, 69
- prepare_features_tables, 51
- prepare_features_tables(), 42, 43, 45, 46, 48–50, 53, 55–59, 61, 62, 64–66, 69
- prepare_libraries_rt, 52
- prepare_libraries_rt(), 42, 43, 45, 46, 48–50, 52, 55–59, 61, 62, 64–66, 69
- prepare_libraries_sop_big, 54
- prepare_libraries_sop_big(), 42, 43, 45, 46, 48–50, 52, 53, 56–59, 61, 62, 64–66, 69

prepare_libraries_sop_closed, 55
prepare_libraries_sop_closed(), 42, 43,
45, 46, 48–50, 52, 53, 55, 57–59, 61,
62, 64–66, 69
prepare_libraries_sop_ecmdb, 56
prepare_libraries_sop_ecmdb(), 42, 43,
45, 46, 48–50, 52, 53, 55, 56, 58, 59,
61, 62, 64–66, 69
prepare_libraries_sop_hmdb, 57
prepare_libraries_sop_hmdb(), 42, 43, 45,
46, 48–50, 52, 53, 55–57, 59, 61, 62,
64–66, 69
prepare_libraries_sop_lotus, 58
prepare_libraries_sop_lotus(), 42, 43,
45, 46, 48–50, 52, 53, 55–58, 61, 62,
64–66, 69
prepare_libraries_sop_merged, 59
prepare_libraries_sop_merged(), 42, 43,
45, 46, 48–50, 52, 53, 55–59, 62,
64–66, 69
prepare_libraries_sop_pubchemlite, 61
prepare_libraries_sop_pubchemlite(),
42, 43, 45, 46, 48–50, 52, 53, 55–59,
61, 64–66, 69
prepare_libraries_spectra, 62
prepare_libraries_spectra(), 42, 43, 45,
46, 48–50, 52, 53, 55–59, 61, 62, 65,
66, 69
prepare_params, 64
prepare_params(), 42, 43, 45, 46, 48–50, 52,
53, 55–59, 61, 62, 64, 66, 69
prepare_taxa, 65
prepare_taxa(), 42, 43, 45, 46, 48–50, 52,
53, 55–59, 61, 62, 64, 65, 69
process_smiles, 67

read_mztab, 68
read_mztab(), 42, 43, 45, 46, 48–50, 52, 53,
55–59, 61, 62, 64–66
run_app, 70
run_app(), 18, 22, 24, 25, 35, 38, 39, 71, 73,
75
run_tima, 71, 72, 73
run_tima(), 18, 22, 24, 25, 35, 38, 39, 70, 73,
75

summarize_results, 20

tima_full, 72
tima_full(), 18, 22, 24, 25, 35, 38, 39, 70,
71, 75
transform_score_sirius_csi, 73

validate_inputs, 75
validate_inputs(), 18, 22, 24, 25, 35, 38,
39, 70, 71, 73

weight_annotations, 76
weight_annotations(), 9, 11, 26, 84
weight_chemo, 20
write_mztab, 82
write_mztab(), 9, 11, 26, 80