Package 'patRoon'

November 12, 2025

Type Package

Title Workflows for Mass-Spectrometry Based Non-Target Analysis

Version 2.3.4

Description Provides an easy-to-use interface to a mass spectrometry based non-target analysis workflow. Various (open-source) tools are combined which provide algorithms for extraction and grouping of features, extraction of MS and MS/MS data, automatic formula and compound annotation and grouping related features to components. In addition, various tools are provided for e.g. data preparation and cleanup, plotting results and automatic reporting.

License GPL-3

LazyData TRUE

RoxygenNote 7.3.2

URL https://github.com/rickhelmus/patRoon

BugReports https://github.com/rickhelmus/patRoon/issues

Encoding UTF-8

Depends R (>= 3.5.0)

SystemRequirements GNU make

Imports methods,
checkmate (>= 1.8.5),
Rcpp,
VennDiagram,
UpSetR,
stats,
utils,
parallel,
grid,
graphics,
RColorBrewer,
data.table,
withr,
digest,

DBI,
RSQLite (>= 2.2.4),
fst,
processx,
tools,
MSnbase,
Biobase,
BiocParallel,
xcms,
cluster,
fastcluster,
gplots,
heatmaply,
dynamicTreeCut,
dendextend,
igraph,
visNetwork,
rJava,
rcdk,
fingerprint,
mzR,
circlize,
miniUI,
rhandsontable,
rstudioapi,
htmlwidgets,
shiny,
shinyjs,
CAMERA,
enviPat,
knitr,
rmarkdown,
flexdashboard,
DT,
bslib (>= 0.4.2),
reactable (>= 0.4.1),
magrittr,
kableExtra,
R.utils,
magick,
glue,
jsonlite,
Rdpack,
rsm,
future,
future.apply,
fs,
yaml,

keys,
backports,
httr,
getPass

Suggests RDCOMClient,
metfRag,
enviPick,
nontarget,
RAMClustR,
cliqueMS,
KPIC,
MetaClean,
MetaCleanData,
testthat,
rlang,
vdiffr,
patRoonData,
devtools,
covr,
DiagrammeR,
DiagrammeRsvg,
rsvg,
pkgload,
splashR,
MS2Quant,
MS2Tox

LinkingTo Rcpp,
rapidjsonr

Config/Needs/pdeps RDCOMClient,
enviPick,
nontarget,
InterpretMSSpectrum,
RAMClustR,
KPIC,
cliqueMS,
MetaClean,
splashR

RdMacros Rdpack

Collate 'RcppExports.R'
'generics.R'
'cache.R'
'main.R'
'workflow-step.R'
'TP.R'
'TP-structure.R'
'TP-CTS.R'
'TP-biotransformer.R'

'TP-formula.R'
'TP-library.R'
'TP-library_formula.R'
'features.R'
'workflow-step-set.R'
'features-set.R'
'feature_groups.R'
'feature_groups-set.R'
'TP-logic.R'
'utils.R'
'utils-adduct.R'
'adduct.R'
'analysisInfo.R'
'check_ui.R'
'utils-components.R'
'components.R'
'check_components.R'
'check_features.R'
'components-camera.R'
'components-features.R'
'components-cliquems.R'
'components-clust.R'
'components-intclust.R'
'components-set.R'
'components-nontarget.R'
'components-nontarget-set.R'
'components-openms.R'
'components-ramclustr.R'
'components-specclust.R'
'feature_annotations.R'
'formulas.R'
'mspeaklists.R'
'compounds.R'
'utils-compounds.R'
'utils-screening.R'
'feature_groups-screening.R'
'feature_groups-screening-set.R'
'components-tps.R'
'compounds-cluster.R'
'mslibrary.R'
'compounds-library.R'
'compounds-metfrag.R'
'utils-feat_annotations-set.R'
'compounds-set.R'
'utils-sirius.R'
'compounds-sirius.R'
'convert.R'
'defunct.R'

'deprecated.R'
'utils-IPO.R'
'doe-optimizer.R'
'feature_groups-bruker.R'
'feature_groups-comparison.R'
'feature_groups-envimass.R'
'feature_groups-filter.R'
'feature_groups-kpic2.R'
'feature_groups-openms.R'
'feature_groups-optimize.R'
'feature_groups-optimize-kpic2.R'
'feature_groups-optimize-openms.R'
'feature_groups-optimize-xcms.R'
'feature_groups-optimize-xcms3.R'
'feature_groups-plot.R'
'feature_groups-sirius.R'
'feature_groups-tasq.R'
'feature_groups-xcms.R'
'feature_groups-xcms3.R'
'utils-bruker.R'
'features-bruker.R'
'features-envipick.R'
'features-kpic2.R'
'features-openms.R'
'features-optimize.R'
'features-optimize-envipick.R'
'features-optimize-kpic2.R'
'features-optimize-openms.R'
'features-optimize-xcms.R'
'features-optimize-xcms3.R'
'features-safd.R'
'features-sirius.R'
'features-tasq.R'
'features-xcms.R'
'features-xcms3.R'
'formulas-bruker.R'
'formulas-genform.R'
'formulas-set.R'
'formulas-sirius.R'
'mslibrary-json.R'
'mslibrary-msp.R'
'mspeaklists-bruker.R'
'utils-mzr.R'
'mspeaklists-mzr.R'
'mspeaklists-set.R'
'multi-process-classic.R'
'multi-process-future.R'
'multi-process.R'

'project-tool.R'
'report-html.R'
'report-html-TPs.R'
'report-html-components.R'
'report-html-feat_annotations.R'
'report-html-features.R'
'report-html-utils.R'
'report-legacy.R'
'report.R'
'utils-TPs.R'
'utils-checkmate.R'
'utils-exported.R'
'utils-feat_annotations.R'
'utils-features.R'
'utils-formulas.R'
'utils-mol.R'
'utils-mslibrary.R'
'utils-mspeaklists.R'
'utils-optimize.R'
'utils-plot.R'
'utils-progress.R'
'utils-sets.R'
'utils-xcms.R'
'zzz.R'

VignetteBuilder knitr

Contents

`patRoon-package` *Workflow solutions for mass-spectrometry based non-target analysis.*

Description

Provides an easy-to-use interface to a mass spectrometry based non-target analysis workflow. Various (open-source) tools are combined which provide algorithms for extraction and grouping of features, extraction of MS and MS/MS data, automatic formula and compound annotation and grouping related features to components. In addition, various tools are provided for e.g. data preparation and cleanup, plotting results and automatic reporting.

Package options

The following package options (see `options`) can be set:

- `patRoon.checkCentroided`: If set to `TRUE` (the default) then the analyses files are verified to be centroided before loading any MS data. While these checks are optimized and cached, it may be useful to set this option to `FALSE` when processing very large numbers of analyses.
- `patRoon.cache.mode`: A `character` setting the current caching mode: `"save"` and `"load"` will only save/load results to/from the cache, `"both"` (default) will do both and `"none"` to completely disable caching. This option can be changed anytime, which might be useful, for instance, to temporarily disable cached results before running a function.
- `patRoon.cache.fileName`: a `character` specifying the name of the cache file (default is '`cache.sqlite`').
- `patRoon.cache.maxEntries`: a `numeric` specifying the maximum number of entries per cache category (default is 100000). When this limit is exceeded, the oldest entries are automatically removed.
- `patRoon.MP.maxProcs`: The maximum number of processes that should be initiated in parallel. A good starting point is the number of physical cores, which is the default as detected by `detectCores`. This option is only used when '`patRoon.MP.method="classic"`'.
- `patRoon.MP.method`: Either `"classic"` or `"future"`. The former is the default and uses **processx** to execute multiple commands in parallel. When `"future"` the `future.apply` package is used for parallelization, which is especially useful for *e.g.* cluster computing.
- `patRoon.MP.futureSched`: Sets the `future.scheduling` function argument for `future_lapply`. Only used if '`patRoon.MP.method="future"`'.
- `patRoon.MP.logPath`: The path used for logging of output from commands executed by multiprocess. Set to `FALSE` to disable logging.

- patRoon.path.pwiz: The path in which the ProteoWizard binaries are installed. If unset an attempt is made to find this directory from the Windows registry and 'PATH' environment variable.
- patRoon.path.GenForm: The path to the GenForm executable. If not set (the default) the internal GenForm binary is used. Only set if you want to override the executable.
- patRoon.path.MetFragCL: The complete file path to the MetFrag CL 'jar' to be used by generateCompoundsMetFrag. Example: "C:/MetFrag2.4.2-CL.jar".
- patRoon.path.MetFragCompTox: The complete file path to the CompTox database 'csv' file. See generateCompounds for more details.
- patRoon.path.MetFragPubChemLite: The complete file path to the PubChemLite database 'csv' file. See generateCompounds for more details.
- patRoon.path.SIRIUS: The directory in which the SIRIUS binaries are installed. Used by all functions that interface with SIRIUS, such as generateFormulasSIRIUS and generateCompoundsSIRIUS. Example: "C:/sirius-win64-3.5.1". Note that the location of the binaries differs for each operating system. are installed in different subdirectories for each location inside this differs for each operating system
- patRoon.path.OpenMS: The path in which the OpenMS binaries are installed.
- patRoon.path.pngquant: The path of the pngquant binary that is used when optimizing '.png' plots generated by reportHTML (with optimizePng set to TRUE). If the binary can be located through the 'PATH' environment variable this option can remain empty. Note that some of the functionality of reportHTML only locates the binary through the 'PATH' environment variable, hence, it is recommended to set up 'PATH' instead.
- patRoon.path.obabel: The path in which the OpenBabel binaries are installed.
- patRoon.path.BiotransFormer The full file path to the biotransformer '.jar' command line utility. This needs to be set when generateTPsBioTransformer is used. For more details see https://bitbucket.org/djoumbou/biotransformer/src/master.

Most external dependencies are provided by **patRoonExt** or otherwise found in the system environment 'PATH' variable. However, the patRoon.path.* options should be set if this fails or you want to override the location. The verifyDependencies function can be used to assess if dependencies are found.

Author(s)

Maintainer: Rick Helmus <r.helmus@uva.nl> (ORCID)

Other contributors:

- Olaf Brock (ORCID) [contributor]
- Vittorio Albergamo (ORCID) [contributor]
- Andrea Brunner (ORCID) [contributor]
- Emma Schymanski (ORCID) [contributor]
- Bas van de Velde (ORCID) [contributor]
- Leon Saal (ORCID) [contributor]

See Also

Useful links:

- https://github.com/rickhelmus/patRoon
- Report bugs at https://github.com/rickhelmus/patRoon/issues

---

addFormulaScoring *Compound annotations class*

Description

Contains data for compound annotations for feature groups.

Usage

```
addFormulaScoring(
  compounds,
  formulas,
  updateScore = FALSE,
  formulaScoreWeight = 1
)

## S4 method for signature 'compounds'
defaultExclNormScores(obj)

## S4 method for signature 'compounds'
show(object)

## S4 method for signature 'compounds'
identifiers(compounds)

## S4 method for signature 'compounds'
filter(
  obj,
  minExplainedPeaks = NULL,
  minScore = NULL,
  minFragScore = NULL,
  minFormulaScore = NULL,
  scoreLimits = NULL,
  ...
)

## S4 method for signature 'compounds'
addFormulaScoring(
  compounds,
  formulas,
```

```
  updateScore = FALSE,
  formulaScoreWeight = 1
)

## S4 method for signature 'compounds'
getMCS(obj, index, groupName)

## S4 method for signature 'compounds'
plotStructure(obj, index, groupName, width = 500, height = 500)

## S4 method for signature 'compounds'
plotScores(
  obj,
  index,
  groupName,
  normalizeScores = "max",
  excludeNormScores = defaultExclNormScores(obj),
  onlyUsed = TRUE
)

## S4 method for signature 'compounds'
annotatedPeakList(
  obj,
  index,
  groupName,
  MSPeakLists,
  formulas = NULL,
  onlyAnnotated = FALSE
)

## S4 method for signature 'compounds'
plotSpectrum(
  obj,
  index,
  groupName,
  MSPeakLists,
  formulas = NULL,
  plotStruct = FALSE,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  maxMolSize = c(0.2, 0.4),
  molRes = c(100, 100),
  ...
)
```

```
## S4 method for signature 'compounds'
consensus(
  obj,
  ...,
  absMinAbundance = NULL,
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE,
  rankWeights = 1,
  labels = NULL
)

## S4 method for signature 'compoundsSet'
show(object)

## S4 method for signature 'compoundsSet'
delete(obj, i, j, ...)

## S4 method for signature 'compoundsSet,ANY,missing,missing'
x[i, j, ..., sets = NULL, updateConsensus = FALSE, drop = TRUE]

## S4 method for signature 'compoundsSet'
filter(obj, ..., sets = NULL, updateConsensus = FALSE, negate = FALSE)

## S4 method for signature 'compoundsSet'
plotSpectrum(
  obj,
  index,
  groupName,
  MSPeakLists,
  formulas = NULL,
  plotStruct = FALSE,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  maxMolSize = c(0.2, 0.4),
  molRes = c(100, 100),
  perSet = TRUE,
  mirror = TRUE,
  ...
)

## S4 method for signature 'compoundsSet'
addFormulaScoring(
  compounds,
  formulas,
```

```
  updateScore = FALSE,
  formulaScoreWeight = 1
)

## S4 method for signature 'compoundsSet'
annotatedPeakList(obj, index, groupName, MSPeakLists, formulas = NULL, ...)

## S4 method for signature 'compoundsSet'
consensus(
  obj,
  ...,
  absMinAbundance = NULL,
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE,
  rankWeights = 1,
  labels = NULL,
  filterSets = FALSE,
  setThreshold = 0,
  setThresholdAnn = 0,
  setAvgSpecificScores = FALSE
)

## S4 method for signature 'compoundsSet'
unset(obj, set)

## S4 method for signature 'compoundsConsensusSet'
unset(obj, set)

## S4 method for signature 'compoundsSIRIUS'
delete(obj, i = NULL, j = NULL, ...)
```

Arguments

`formulas` The `formulas` object that should be used for scoring/annotation. For `plotSpectrum` and `annotatedPeakList`: set to `NULL` to ignore.

`updateScore`, `formulaScoreWeight`
If `updateScore=TRUE` then the annotation `score` column is updated by adding normalized values of the formula score (weighted by '`formulaScoreWeight`'). Currently, this **only** makes sense for annotations performed with `MetFrag`!

`obj`, `object`, `compounds`, `x`
The `compound` object.

`minExplainedPeaks`, `scoreLimits`
Passed to the `featureAnnotations` method.

`minScore`, `minFragScore`, `minFormulaScore`
Minimum overall score, in-silico fragmentation score and formula score, respectively. Set to `NULL` to ignore. The `scoreLimits` argument allows for more advanced score filtering.

...	For plotSpectrum: Further arguments passed to plot. For delete: passed to the function specified as j. for filter: passed to the featureAnnotations method. For consensus: any further (and unique) compounds objects. For sets workflow methods: further arguments passed to the base compounds method.
index	The numeric index of the candidate structure. For plotStructure and getMCS: multiple indices (*i.e.* vector with length >=2) should be specified to plot/calculate the most common substructure (MCS). Alternatively, '-1' may be specified to select all candidates. For plotSpectrum: two indices can be specified to compare spectra. In this case groupName should specify values for the spectra to compare.
groupName	The name of the feature group (or feature groups when comparing spectra) to which the candidate belongs.
width, height	The dimensions (in pixels) of the raster image that should be plotted.
normalizeScores	A character that specifies how normalization of annotation scorings occurs. Either "none" (no normalization), "max" (normalize to max value) or "minmax" (perform min-max normalization). Note that normalization of negative scores (e.g. output by SIRIUS) is always performed as min-max. Furthermore, currently normalization for compounds takes the original min/max scoring values into account when candidates were generated. Thus, for compounds scoring, normalization is not affected when candidate results were removed after they were generated (*e.g.* by use of filter).
excludeNormScores	A character vector specifying any compound scoring names that should *not* be normalized. Set to NULL to normalize all scorings. Note that whether any normalization occurs is set by the excludeNormScores argument. For compounds: By default score and individualMoNAScore are set to mimic the behavior of the MetFrag web interface.
onlyUsed	If TRUE then only scorings are plotted that actually have been used to rank data (see the scoreTypes argument to generateCompoundsMetFrag for more details).
MSPeakLists	The MSPeakLists object that was used to generate the candidate
onlyAnnotated	Set to TRUE to filter out any peaks that could not be annotated.
plotStruct	If TRUE then the candidate structure is drawn in the spectrum. Currently not supported when comparing spectra.
title	The title of the plot. If NULL a title will be automatically made.
specSimParams	A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.
mincex	The formula annotation labels are automatically scaled. The mincex argument forces a minimum cex value for readability.
xlim, ylim	Sets the plot size limits used by plot. Set to NULL for automatic plot sizing.

`maxMolSize` Numeric vector of size two with the maximum width/height of the candidate structure (relative to the plot size).

`molRes` Numeric vector of size two with the resolution of the candidate structure (in pixels).

`absMinAbundance`, `relMinAbundance`
Minimum absolute or relative ('`0-1`') abundance across objects for a result to be kept. For instance, `relMinAbundance=0.5` means that a result should be present in at least half of the number of compared objects. Set to '`NULL`' to ignore and keep all results. Limits cannot be set when `uniqueFrom` is not `NULL`.

`uniqueFrom` Set this argument to only retain compounds that are unique within one or more of the objects for which the consensus is made. Selection is done by setting the value of `uniqueFrom` to a `logical` (values are recycled), `numeric` (select by index) or a `character` (as obtained with `algorithm(obj)`). For `logical` and `numeric` values the order corresponds to the order of the objects given for the consensus. Set to `NULL` to ignore.

`uniqueOuter` If `uniqueFrom` is not `NULL` and if `uniqueOuter=TRUE`: only retain data that are also unique between objects specified in `uniqueFrom`.

`rankWeights` A numeric vector with weights of to calculate the mean ranking score for each candidate. The value will be re-cycled if necessary, hence, the default value of '`1`' means equal weights for all considered objects.

`labels` A `character` with names to use for labelling. If `NULL` labels are automatically generated.

`i`, `j`, `drop` Passed to the `featureAnnotations` method.

`sets` (**sets workflow**) A `character` with name(s) of the sets to keep (or remove if `negate=TRUE`). Note: if `updateConsensus=FALSE` then the `setCoverage` column of the annotation results is not updated.

`updateConsensus`
(**sets workflow**) If `TRUE` then the annonation consensus among set results is updated. See the `Sets workflows` section for more details.

`negate` Passed to the `featureAnnotations` method.

`perSet`, `mirror` (**sets workflow**) If `perSet=TRUE` then the set specific mass peaks are annotated separately. Furthermore, if `mirror=TRUE` (and there are two sets in the object) then a mirror plot is generated.

`filterSets` (**sets workflow**) Controls how algorithms concensus abundance filters are applied. See the `Sets workflows` section below.

`setThreshold`, `setThresholdAnn`
(**sets workflow**) Thresholds used to create the annotation set consensus. See `generateCompounds`.

`setAvgSpecificScores`
(**sets workflow**) If `TRUE` then set specific annotation scores (*e.g.* MS/MS and isotopic pattern match scores) are averaged for the set consensus. See `generateCompounds`.

`set` (**sets workflow**) The name of the set.

Details

compounds objects are obtained from compound generators. This class is derived from the featureAnnotations class, please see its documentation for more methods and other details.

Value

addFormulaScoring returns a compounds object updated with formula scoring.

getMCS returns an **rcdk** molecule object (IAtomContainer).

consensus returns a compounds object that is produced by merging multiple specified compounds objects.

Methods (by generic)

- defaultExclNormScores(compounds): Returns default scorings that are excluded from normalization.
- show(compounds): Show summary information for this object.
- identifiers(compounds): Returns a list containing for each feature group a character vector with database identifiers for all candidate compounds. The list is named by feature group names, and is typically used with the identifiers option of generateCompoundsMetFrag.
- filter(compounds): Provides rule based filtering for generated compounds. Useful to eliminate unlikely candidates and speed up further processing. Also see the featureAnnotations method.
- addFormulaScoring(compounds): Adds formula ranking data from a formulas object as an extra compound candidate scoring (formulaScore column). The formula score for each compound candidate is between '0-1', where *zero* means no match with any formula candidates, and *one* means that the compound candidate's formula is the highest ranked.
- getMCS(compounds): Calculates the maximum common substructure (MCS) for two or more candidate structures for a feature group. This method uses the get.mcs function from **rcdk**.
- plotStructure(compounds): Plots a structure of a candidate compound using the **rcdk** package. If multiple candidates are specified (*i.e.* by specifying a vector for index) then the maximum common substructure (MCS) of the selected candidates is drawn.
- plotScores(compounds): Plots a barplot with scoring of a candidate compound.
- annotatedPeakList(compounds): Returns an MS/MS peak list annotated with data from a given candidate compound for a feature group.
- plotSpectrum(compounds): Plots an annotated spectrum for a given candidate compound for a feature group. Two spectra can be compared by specifying a two-sized vector for the index and groupName arguments.
- consensus(compounds): Generates a consensus of results from multiple objects. In order to rank the consensus candidates, first each of the candidates are scored based on their original ranking (the scores are normalized and the highest ranked candidate gets value '1'). The (weighted) mean is then calculated for all scorings of each candidate to derive the final ranking (if an object lacks the candidate its score will be '0'). The original rankings for each object is stored in the rank columns.

Slots

`MS2QuantMeta` Metadata from **MS2Quant** filled in by `predictRespFactors`.

`setThreshold,setThresholdAnn,setAvgSpecificScores` (**sets workflow**) A copy of the equally named arguments that were passed when this object was created by `generateCompounds`.

`origFGNames` (**sets workflow**) The original (order of) names of the `featureGroups` object that was used to create this object.

S4 class hierarchy

- `featureAnnotations`
  - `compounds`
    - `compoundsConsensus`
    - `compoundsMF`
    - `compoundsSet`
      - `compoundsConsensusSet`
    - `compoundsUnset`
    - `compoundsSIRIUS`

Source

Subscripting of formulae for plots generated by `plotSpectrum` is based on the `chemistry2expression` function from the ReSOLUTION package.

Sets workflows

The `compoundsSet` class is applicable for sets workflows. This class is derived from `compounds` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class `workflowStepSet`.
- `unset` Converts the object data for a specified set into a 'non-set' object (`compoundsUnset`), which allows it to be used in 'regular' workflows. Only the annotation results that are present in the specified set are kept (based on the set consensus, see below for implications).

The following methods are changed or with new functionality:

- `filter` and the subset operator (`[`) Can be used to select data that is only present for selected sets. Depending on the `updateConsenus`, both either operate on set consensus or original data (see below for implications).
- `annotatedPeakList` Returns a combined annotation table with all sets.
- `plotSpectrum` Is able to highlight set specific mass peaks (`perSet` and `mirror` arguments).
- `consensus` Creates the algorithm consensus based on the original annotation data (see below for implications). Then, like the sets workflow method for `generateCompounds`, a consensus is made for all sets, which can be controlled with the `setThreshold` and `setThresholdAnn` arguments. The candidate coverage among the different algorithms is calculated for each set (*e.g.* `coverage-positive` column) and for all sets (`coverage` column), which is based on the

presence of a candidate in all the algorithms from all sets data. The `consensus` method for sets workflow data supports the `filterSets` argument. This controls how the algorithm consensus abundance filters (`absMinAbundance`/`relMinAbundance`) are applied: if `filterSets=TRUE` then the minimum of all `coverage` set specific columns is used to obtain the algorithm abundance. Otherwise the overall `coverage` column is used. For instance, consider a consensus object to be generated from two objects generated by different algorithms (*e.g.* `SIRIUS` and `MetFrag`), which both have a positive and negative set. Then, if a candidate occurs with both algorithms for the positive mode set, but only with the first algorithm in the negative mode set, `relMinAbundance=1` will remove the candidate if `filterSets=TRUE` (because the minimum relative algorithm abundance is '`0.5`'), while `filterSets=FALSE` will not remove the candidate (because based on all sets data the candidate occurs in both algorithms).

- `addFormulaScoring` Adds the formula scorings to the original data and re-creates the annotation set consensus (see below for implications).

Two types of annotation data are stored in a `compoundsSet` object:

1. Annotations that are produced from a consensus between set results (see `generateCompounds`).
2. The 'original' annotation data per set, prior to when the set consensus was made. This includes candidates that were filtered out because of the thresholds set by `setThreshold` and `setThresholdAnn`. However, when `filter` or subsetting (`[`) operations are performed, the original data is also updated.

In most cases the first data is used. However, in a few cases the original annotation data is used (as indicated above), for instance, to re-create the set consensus. It is important to realize that the original annotation data may have *additional* candidates, and a newly created set consensus may therefore have 'new' candidates. For instance, when the object consists of the sets `"positive"` and `"negative"` and `setThreshold=1` was used to create it, then `compounds[, sets = "positive", updateConsensus = TRUE]` may now have additional candidates, *i.e.* those that were not present in the `"negative"` set and were previously removed due to the consensus threshold filter.

Note

The values ranges in the `scoreLimits` slot, which are used for normalization of scores, are based on the *original* scorings when the compounds were generated (*prior* to employing the `topMost` filter to `generateCompounds`).

References

Guha R (2007). "Chemical Informatics Functionality in R." *Journal of Statistical Software*, **18**(6).

See Also

The `featureAnnotations` base class for more relevant methods and `generateCompounds`.

adduct-class *Generic adduct class*

Description

Objects from this class are used to specify adduct information in an algorithm independent way.

Usage

```
adduct(...)

## S4 method for signature 'adduct'
show(object)

## S4 method for signature 'adduct'
as.character(x, format = "generic", err = TRUE)
```

Arguments

x, object	An adduct object.
format	A character that specifies the source format. "generic" is an internally used generic format that supports full textual conversion. Examples: "[M+H]+", "[2M+H]+", "[M+3H]3+". "sirius" Is the format used by SIRIUS. It is similar to generic but does not allow multiple charges/molecules. See the SIRIUS manual for more details. "genform" and "metfrag" support fixed types of adducts which can be obtained with the GenFormAdducts and MetFragAdducts functions, respectively. "openms" is the format used by the MetaboliteAdductDecharger tool. "cliquems" is the format used by **cliqueMS**.
err	If TRUE then an error will be thrown if conversion fails, otherwise returns without data.
...	Any of add, sub, molMult and/or charge. See Slots.

Methods (by generic)

- show(adduct): Shows summary information for this object.
- as.character(adduct): Converts an adduct object to a specified character format.

Slots

add,sub A character with one or more formulas to add/subtract.

molMult How many times the original molecule is present in this molecule (*e.g.* for a dimer this would be '2'). Default is '1'.

charge The final charge of the adduct (default '1').

See Also

as.adduct for easy creation of adduct objects and adduct utilities for other adduct functionality.

Examples

```
adduct("H") # [M+H]+
adduct(sub = "H", charge = -1) # [M-H]-
adduct(add = "K", sub = "H2", charge = -1) # [M+K-H2]+
adduct(add = "H3", charge = 3) # [M+H3]3+
adduct(add = "H", molMult = 2) # [2M+H]+

as.character(adduct("H")) # returns "[M+H]+"
```

---

adduct-utils *Adduct utilities*

Description

Several utility functions to work with adducts.

Usage

```
GenFormAdducts()

MetFragAdducts()

as.adduct(x, format = "generic", isPositive = NULL, charge = NULL, err = TRUE)

calculateIonFormula(formula, adduct)

calculateNeutralFormula(formula, adduct)
```

Arguments

x	The object that should be converted. Should be a character string, a numeric MetFrag adduct identifier (adduct_mode column obtained with MetFragAdducts) or an adduct object (in which case no conversion occurs).
format	A character that specifies the source format. "generic" is an internally used generic format that supports full textual conversion. Examples: "[M+H]+", "[2M+H]+", "[M+3H]3+". "sirius" Is the format used by SIRIUS. It is similar to generic but does not allow multiple charges/molecules. See the SIRIUS manual for more details. "genform" and "metfrag" support fixed types of adducts which can be obtained with the GenFormAdducts and MetFragAdducts functions, respectively. "openms" is the format used by the MetaboliteAdductDecharger tool. "cliquems" is the format used by **cliqueMS**.

isPositive	A logical that specifies whether the adduct should be positive. Should only be set when format="metfrag" and x is a numeric identifier.
charge	The final charge. Only needs to be set when format="openms".
err	If TRUE then an error will be thrown if conversion fails, otherwise returns without data.
formula	A character vector with formulae to convert.
adduct	An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+".

Details

GenFormAdducts returns a table with information on adducts supported by GenForm.

MetFragAdducts returns a table with information on adducts supported by MetFrag.

as.adduct Converts an object in to an adduct object.

calculateIonFormula Converts one or more neutral formulae to adduct ions.

calculateNeutralFormula Converts one or more adduct ions to neutral formulae.

Examples

```
as.adduct("[M+H]+")
as.adduct("[M+H2]2+")
as.adduct("[2M+H]+")
as.adduct("[M-H]-")
as.adduct("+H", format = "genform")
as.adduct(1, isPositive = TRUE, format = "metfrag") # MetFrag adduct ID 1 --> returns [M+H]+

calculateIonFormula("C2H4O", "[M+H]+") # C2H5O
calculateNeutralFormula("C2H5O", "[M+H]+") # C2H4O
```

---

analysis-information *Properties of sample analyses*

Description

Properties for the sample analyses used in the workflow and utilities to automatically generate this information.

Usage

```
generateAnalysisInfo(
  paths,
  groups = "",
  blanks = "",
  concs = NULL,
  norm_concs = NULL,
```

```
  formats = MSFileFormats()
)

generateAnalysisInfoFromEnviMass(path)
```

Arguments

paths	A character vector containing one or more file paths that should be used for finding the analyses.
groups, blanks	An (optional) character vector containing replicate groups and blanks, respectively (will be recycled). If groups is an empty character string ("") the analysis name will be set as replicate group.
concs	An optional numeric vector containing concentration values for each analysis. Can be NA if unknown. If the length of concs is less than the number of analyses the remainders will be set to NA. Set to NULL to not include concentration data.
norm_concs	An optional numeric vector containing concentrations used for *feature normalization* (see the Feature intensity normalization section in the featureGroups documentation). NA values are allowed for analyses that should not be normalized (*e.g.* because no IS is present). If the length of norm_concs is less than the number of analyses the remainders will be set to NA. Set to NULL to not include normalization concentration data.
formats	A character vector of analyses file types to consider. Analyses not present in these formats will be ignored. For valid values see MSFileFormats.
path	The path of the enviMass project.

Details

In **patRoon** a *sample analysis*, or simply *analysis*, refers to a single MS analysis file (sometimes also called *sample* or *file*). The *analysis information* summarizes several properties for the analyses, and is used in various steps throughout the workflow, such as findFeatures, averaging intensities of feature groups and blank subtraction. This information should be in a data.frame, with the following columns:

- path the full path to the directory of the analysis.
- analysis the file name **without** extension. Must be **unique**, even if the path is different.
- group name of *replicate group*. A replicate group is used to group analyses together that are replicates of each other. Thus, the group column for all analyses considered to be belonging to the same replicate group should have an equal (but unique) value. Used for *e.g.* averaging and filter.
- blank all analyses within this replicate group are used by the featureGroups method of filter for blank subtraction. Multiple entries can be entered by separation with a comma.
- conc a numeric value specifying the 'concentration' for the analysis. This can be actually any kind of numeric value such as exposure time, dilution factor or anything else which may be used to form a linear relationship.
- norm_conc a numeric value specifying the *normalization concentration* for the analysis. See the Feature intensity normalization section in the featureGroups documentation) for more details.

Most workflows steps work with 'mzXML' and 'mzML' file formats. However, some algorithms only support support one format (*e.g.* findFeaturesOpenMS, findFeaturesEnviPick) or a proprietary format (findFeaturesBruker). To mix such algorithms in the same workflow, the analyses should be present in all required formats within the *same* directory as specified by the path column.

Each analysis should only be specified *once* in the analysis information, even if multiple file formats are available. The path and analysis columns are internally used by **patRoon** to automatically find the path of analysis files with the required format.

The group column is *mandatory* and needs to be non-empty for each analysis. The blank column should also be present, however, this may be empty ("") for analyses where no blank subtraction should occur. The conc column is only required when obtaining regression information is desired with the as.data.table method. Similarly, the norm_conc is only necessary for the normInts method.

generateAnalysisInfo is an utility function that automatically generates a data.frame with analysis information. It scans the directories specified from the paths argument for analyses, and uses this to automatically fill in the analysis and path columns. Furthermore, this function also correctly handles analyses which are available in multiple formats.

generateAnalysisInfoFromEnviMass loads analysis information from an **enviMass** project. Note: this funtionality has only been tested with older versions of **enviMass**.

---

analysisinfo-dataframe

AnalysisInfo data.frame methods

Description

Various parsing and plotting functions for the analysisInfo data.frame.

Usage

```
## S4 method for signature 'data.frame'
getTICs(obj, retentionRange = NULL, MSLevel = 1)

## S4 method for signature 'data.frame'
getBPCs(obj, retentionRange = NULL, MSLevel = 1)

## S4 method for signature 'data.frame'
plotTICs(
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  colourBy = c("none", "analyses", "rGroups"),
  showLegend = TRUE,
  xlim = NULL,
  ylim = NULL,
```

```
  ...
)

## S4 method for signature 'data.frame'
plotBPCs(
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  colourBy = c("none", "analyses", "rGroups"),
  showLegend = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)
```

Arguments

obj	An analysisInfo data.frame object as obtained by generateAnalysisInfo function.
retentionRange	Range of retention time (in seconds), m/z, respectively. Should be a numeric vector with length of two containing the min/max values. The maximum can be Inf to specify no maximum range. Set to NULL to skip this step.
MSLevel	Integer vector with the ms levels (i.e., 1 for MS1 and 2 for MS2) to obtain traces.
retMin	Plot retention time in minutes (instead of seconds).
title	Character string used for title of the plot. If NULL a title will be automatically generated.
colourBy	Sets the automatic colour selection: "none" for a single colour or "analyses"/"rGroups" for a distinct colour per analysis or analysis replicate group.
showLegend	Plot a legend if TRUE.
xlim, ylim	Sets the plot size limits used by plot. Set to NULL for automatic plot sizing.
...	Further arguments passed to plot.

Functions

- getTICs(data.frame): Obtain the total ion chromatogram/s (TICs) of the analyses.
- getBPCs(data.frame): Obtain the base peak chromatogram/s (BPCs) of the analyses.
- plotTICs(data.frame): Plots the TICs of the analyses.
- plotBPCs(data.frame): Plots the BPCs of the analyses.

Author(s)

Ricardo Cunha, <cunha@iuta.de>

bruker-utils *Bruker DataAnalysis utilities*

Description

Miscellaneous utility functions which interface with Bruker DataAnalysis

Usage

```
showDataAnalysis()

setDAMethod(anaInfo, method, close = TRUE)

revertDAAnalyses(anaInfo, close = TRUE, save = close)

recalibrarateDAFiles(anaInfo, close = TRUE, save = close)

getDACalibrationError(anaInfo)

addDAEIC(
  analysis,
  path,
  mz,
  mzWindow = 0.005,
  ctype = "EIC",
  mtype = "MS",
  polarity = "both",
  bgsubtr = FALSE,
  fragpath = "",
  name = NULL,
  hideDA = TRUE,
  close = FALSE,
  save = close
)

addAllDAEICs(
  fGroups,
  mzWindow = 0.005,
  ctype = "EIC",
  bgsubtr = FALSE,
  name = TRUE,
  onlyPresent = TRUE,
  hideDA = TRUE,
  close = FALSE,
  save = close
)
```

Arguments

anaInfo	Analysis info table
method	The full path of the DataAnalysis method.
close, save	If TRUE then Bruker files are closed and saved after processing with DataAnalysis, respectively. Setting close=TRUE prevents that many analyses might be opened simultaneously in DataAnalysis, which otherwise may use excessive memory or become slow. By default save is TRUE when close is TRUE, which is likely what you want as otherwise any processed data is lost.
analysis	Analysis name (without file extension).
path	path of the analysis.
mz	*m/z* (Da) value used for the chromatographic trace (if applicable).
mzWindow	*m/z* window (in Da) used for the chromatographic trace (if applicable).
ctype	Type of the chromatographic trace. Valid options are: "EIC" (extracted ion chromatogram), "TIC" (total ion chromatogram, only for addDAEIC) and "BPC" (Base Peak Chromatogram).
mtype	MS filter for chromatographic trace. Valid values are: "all", "MS", "MSMS", "allMSMS" and "BBCID".
polarity	Polarity filter for chromatographic trace. Valid values: "both", "positive" and "negative".
bgsubtr	If TRUE then background subtraction ('Spectral' algorithm) will be performed.
fragpath	Precursor *m/z* used for MS/MS traces ("" for none).
name	For addDAEIC: the name for the chromatographic trace. For addAllEICs: TRUE to automatically set EIC names. Set to NULL for none.
hideDA	Hides DataAnalysis while adding the chromatographic trace (faster).
fGroups	The featureGroups object for which EICs should be made.
onlyPresent	If TRUE then EICs are only generated for analyses where the feature was detected.

Details

These functions communicate directly with Bruker DataAnalysis to provide various functionality, such as calibrating and exporting data and adding chromatographic traces. For this the **RDCOM-Client** package is required to be installed.

showDataAnalysis makes a hidden DataAnalysis window visible again. Most functions using DataAnalysis will hide the window during processing for efficiency reasons. If the window remains hidden (*e.g.* because there was an error) this function can be used to make it visible again. This function can also be used to start DataAnalysis if it is not running yet.

setDAMethod Sets a given DataAnalysis method ('.m' file) to a set of analyses. **NOTE**: as a workaround for a bug in DataAnalysis, this function will save(!), close and re-open any analyses that are already open prior to setting the new method. The close argument only controls whether the file should be closed after setting the method (files are always saved).

revertDAAnalyses Reverts a given set of analyses to their unprocessed raw state.

recalibrarateDAFiles Performs automatic mass recalibration of a given set of analyses. The current method settings for each analyses will be used.

getDACalibrationError is used to obtain the standard deviation of the current mass calibration (in ppm).

addDAEIC adds an Extracted Ion Chromatogram (EIC) or other chromatographic trace to a given analysis which can be used directly with DataAnalysis.

addAllDAEICs adds Extracted Ion Chromatograms (EICs) for all features within a featureGroups object.

Value

getDACalibrationError returns a data.frame with a column of all analyses (named analysis) and their mass error (named error).

See Also

analysis-information

---

caching *Utilities for caching of workflow data.*

Description

Several utility functions for caching workflow data. The most important function is clearCache; other functions are primarily for internal use.

Usage

```
makeHash(..., checkDT = TRUE)

makeFileHash(..., length = Inf)

loadCacheData(category, hashes, dbArg = NULL, simplify = TRUE, fixDTs = TRUE)

saveCacheData(category, data, hash, dbArg = NULL)

clearCache(what = NULL, file = NULL, vacuum = TRUE)
```

Arguments

...	Arguments/objects to be used for hashing.
checkDT	logical, set to TRUE with (a list with) data.tables to ensure reproducible hashing. Otherwise can be set to FALSE to improve performance.
length	Maximum file length to hash. Passed to digest.
category	The category of the object to be cached.

hashes	A character with one more hashes (*e.g.* obtained with makeHash) of the objects to be loaded.
dbArg	Alternative connection to database. Default is NULL and uses the cache options as defined by 'patRoon.cache.fileName'. Mainly used internally to improve performance.
simplify	If TRUE and length(hashes)==1 then the returned data is returned directly, otherwise the data is in a list.
fixDTs	Should be TRUE if cached data consists of (nested) data.tables. Otherwise can be FALSE to speed up loading.
data	The object to be cached.
hash	The hash string of the object to be cached (*e.g.* obtained with makeHash).
what	This argument describes what should be done. When what = NULL this function will list which tables are present along with an indication of their size (database rows). If what = "all" then the complete file will be removed. Otherwise, what should be a character string (a regular expression) that is used to match the table names that should be removed.
file	The cache file. If NULL then the value of the patRoon.cache.fileName option is used.
vacuum	If TRUE then the VACUUM operation will be run on the cache database to reduce the file size. Setting this to FALSE might be handy to avoid long processing times on large cache databases.

Details

makeHash Make a hash string of given arguments.

makeFileHash Generates a hash from the contents of one or more files.

loadCacheData Loads cached data from a database.

saveCacheData caches data in a database.

clearCache will either remove one or more tables within the cache sqlite database or simply wipe the whole cache file. Removing tables will VACUUM the database (unless vacuum=FALSE), which may take some time for large cache files.

---

checkFeatures *Interactive GUI utilities to check workflow data*

Description

These functions provide interactive utilities to explore and review workflow data using a **shiny** graphical user interface (GUI). In addition, unsatisfactory data (*e.g.* noise identified as a feature and unrelated feature groups in a component) can easily be selected for removal.

Usage

```
checkFeatures(
  fGroups,
  session = "checked-features.yml",
  EICParams = getDefEICParams(),
  clearSession = FALSE
)

checkComponents(
  components,
  fGroups,
  session = "checked-components.yml",
  EICParams = getDefEICParams(),
  clearSession = FALSE
)

## S4 method for signature 'components'
checkComponents(
  components,
  fGroups,
  session = "checked-components.yml",
  EICParams = getDefEICParams(),
  clearSession = FALSE
)

importCheckFeaturesSession(
  sessionIn,
  sessionOut,
  fGroups,
  rtWindow = 6,
  mzWindow = 0.002,
  overWrite = FALSE
)

## S4 method for signature 'featureGroups'
checkFeatures(
  fGroups,
  session = "checked-features.yml",
  EICParams = getDefEICParams(),
  clearSession = FALSE
)

getMCTrainData(fGroups, session)

predictCheckFeaturesSession(fGroups, session, model = NULL, overWrite = FALSE)
```

Arguments

fGroups	A featureGroups object. This should be the 'new' object for importCheckFeaturesSession for which the session needs to be imported.
session	The session file name.
EICParams	A named list with parameters used for extracted ion chromatogram (EIC) creation. See the EIC parameters documentation for more details.
clearSession	If TRUE the session will be completely cleared before starting the GUI. This effectively removes all selections for data removal.
components	The components to be checked.
sessionIn, sessionOut	The file names for the input and output sessions.
rtWindow	The retention time window (seconds) used to relate 'old' with 'new' feature groups.
mzWindow	The *m/z* window (in Da) used to relate 'old' with 'new' feature groups.
overWrite	Set to TRUE to overwrite the output session file if it already exists. If FALSE, the function will stop with an error message.
model	The model that was created with **MetaClean** and that should be used to predict pass/fail data. If NULL, the example model of the **MetaCleanData** package is used.

Details

The data selected for removal is stored in *sessions*. These are 'YAML' files to allow easy external manipulation. The sessions can be used to restore the selections that were made for data removal when the GUI tool is executed again. Furthermore, functionality is provided to import and export sessions. To actually remove the data the filter method should be used with the session file as input.

checkComponents is used to review components and their feature groups contained within. A typical use case is to verify that peaks from features that were annotated as related adducts and/or isotopes are correctly aligned.

importCheckFeaturesSession is used to import a session file that was generated from a different featureGroups object. This is useful to avoid re-doing manual interpretation of chromatographic peaks when, for instance, feature group data is re-created with different parameters.

checkFeatures is used to review chromatographic information for feature groups. Its main purpose is to assist in reviewing the quality of detected feature (groups) and easily select unwanted data such as features with poor peak shapes or noise.

getMCTrainData converts a session created by checkFeatures to a data.frame that can be used by the **MetaClean** to train a new model. The output format is comparable to that from getPeakQualityMetrics.

predictCheckFeaturesSession Uses ML data from **MetaClean** to predict the quality (Pass/Fail) of feature group data, and converts this to a session which can be reviewed with checkFeatures and used to remove unwanted feature groups by filter.

Value

A dataframe with the class predictions as well as the associated probabilities for each EIC as returned by the `MetaClean::getPredicitons` function. The dataframe has the four columns: EIC, Pred_Class, Pred_Prob_Pass, Pred_Prob_Fail.

Note

The `topMost` and `topMostByRGroup` EIC parameters (`EICParams`) are ignored.

`checkComponents`: Some componentization algorithms (*e.g.* `generateComponentsNontarget` and `generateComponentsTPs`) may output components where the same feature group in a component is present multiple times, for instance, when multiple TPs are matched to the same feature group. If such a feature group is selected for removal, then *all* of its result in the component will be marked for removal.

`getMCTrainData` only uses session data for selected feature groups. Selected features for removal are ignored, as this is not supported by **MetaClean**.

References

Chetnik K, Petrick L, Pandey G (2020). "MetaClean: a machine learning-based classifier for reduced false positive peak detection in untargeted LC-MS metabolomics data." *Metabolomics*, **16**(11). doi:10.1007/s11306020017383.

---

comparison *Comparing feature groups*

Description

Functionality to compare feature groups and make a consensus.

Usage

```
comparison(..., groupAlgo, groupArgs = list(rtalign = FALSE))

## S4 method for signature 'featureGroups'
comparison(..., groupAlgo, groupArgs = list(rtalign = FALSE))

## S4 method for signature 'featureGroupsComparison,missing'
plot(x, retMin = FALSE, ...)

## S4 method for signature 'featureGroupsComparison'
plotVenn(obj, which = NULL, ...)

## S4 method for signature 'featureGroupsComparison'
plotUpSet(obj, which = NULL, ...)

## S4 method for signature 'featureGroupsComparison'
```

```
plotChord(obj, addSelfLinks = FALSE, addRetMzPlots = TRUE, ...)

## S4 method for signature 'featureGroupsComparison'
consensus(
  obj,
  absMinAbundance = NULL,
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE,
  verifyAnaInfo = TRUE
)

## S4 method for signature 'featureGroupsSet'
comparison(..., groupAlgo, groupArgs = list(rtalign = FALSE))

## S4 method for signature 'featureGroupsComparisonSet'
consensus(obj, ...)
```

Arguments

`...`	For `comparison`: `featureGroups` objects that should be compared. If the arguments are named (*e.g.* `myGroups = fGroups`) then these are used for labelling, otherwise objects are automatically labelled by their `algorithm`. For `plot`, `plotVenn`, `plotChord`: further options passed to `plot`, **VennDiagram** plotting functions (*e.g.* `draw.pairwise.venn`) and `chordDiagram` respectively. For `plotUpSet`: any further arguments passed to the `plotUpSet` method defined for `featureGroups`.
`groupAlgo`	The `feature grouping algorithm` that should be used for grouping *pseudo* features (see details). Valid values are: `"xcms"`, `xcms3`, `kpic2` or `"openms"`.
`groupArgs`	A `list` containing further parameters for `feature grouping`.
`x`, `obj`	The `featureGroupsComparison` object.
`retMin`	If `TRUE` retention times are plotted as minutes (seconds otherwise).
`which`	A character vector specifying one or more labels of compared feature groups. For `plotVenn`: if `NULL` then all compared groups are used.
`addSelfLinks`	If `TRUE` then 'self-links' are added which represent non-shared data.
`addRetMzPlots`	Set to `TRUE` to enable *m/z vs* retention time scatter plots.
`absMinAbundance`, `relMinAbundance`	Minimum absolute or relative ('`0-1`') abundance across objects for a result to be kept. For instance, `relMinAbundance=0.5` means that a result should be present in at least half of the number of compared objects. Set to '`NULL`' to ignore and keep all results. Limits cannot be set when `uniqueFrom` is not `NULL`.
`uniqueFrom`	Set this argument to only retain feature groups that are unique within one or more of the objects for which the consensus is made. Selection is done by setting the value of `uniqueFrom` to a `logical` (values are recycled), `numeric` (select by index) or a `character` (as obtained with `algorithm(obj)`). For `logical` and

numeric values the order corresponds to the order of the objects given for the consensus. Set to NULL to ignore.

uniqueOuter If uniqueFrom is not NULL and if uniqueOuter=TRUE: only retain data that are also unique between objects specified in uniqueFrom.

verifyAnaInfo If FALSE then the analysis information is not verified to be equal for all compared objects. This is mainly only useful when the data is the same but stored in different formats (*e.g.* mzXML/mzML).

Details

Feature groups objects originating from differing feature finding and/or grouping algorithms (or their parameters) may be compared to assess their output and generate a consensus.

The comparison method generates a featureGroupsComparison object from given feature groups objects, which in turn may be used for (visually) comparing presence of feature groups and generating a consensus. Internally, this function will collapse each feature groups object to *pseudo* features objects by averaging their retention times, *m/z* values and intensities, where each original feature groups object becomes an 'analysis'. All *pseudo* features are then grouped using regular feature grouping algorithms so that a comparison can be made.

plot generates an *m/z vs* retention time plot.

plotVenn plots a Venn diagram outlining unique and shared feature groups between up to five compared feature groups.

plotUpSet plots an UpSet diagram outlining unique and shared feature groups.

plotChord plots a chord diagram to visualize the distribution of feature groups.

consensus combines all compared feature groups and averages their retention, *m/z* and intensity data. Not yet supported for sets workflows.

Value

comparison returns a featureGroupsComparison object.

plotVenn (invisibly) returns a list with the following fields:

- gList the gList object that was returned by the utilized **VennDiagram** plotting function.
- areas The total area for each plotted group.
- intersectionCounts The number of intersections between groups.

The order for the areas and intersectionCounts fields is the same as the parameter order from the used plotting function (see *e.g.* draw.pairwise.venn and draw.triple.venn).

consensus returns a featureGroups object with a consensus from the compared feature groups.

componentsClust-class *Base class for components that are based on hierarchical clustered data.*

Description

This base class is derived from components and is used to store components resulting from hierarchical clustering information, for instance, generated by generateComponentsIntClust and generateComponentsSpecClust.

Usage

```
## S4 method for signature 'componentsClust'
delete(obj, ...)

## S4 method for signature 'componentsClust'
clusters(obj)

## S4 method for signature 'componentsClust'
cutClusters(obj)

## S4 method for signature 'componentsClust'
clusterProperties(obj)

## S4 method for signature 'componentsClust'
treeCut(obj, k = NULL, h = NULL)

## S4 method for signature 'componentsClust'
treeCutDynamic(obj, maxTreeHeight, deepSplit, minModuleSize)

## S4 method for signature 'componentsClust,missing'
plot(
  x,
  pal = "Paired",
  numericLabels = TRUE,
  colourBranches = length(x) < 50,
  showLegend = length(x) < 20,
  ...
)

## S4 method for signature 'componentsClust'
plotSilhouettes(obj, kSeq, pch = 16, type = "b", ...)
```

Arguments

...	Further options passed to plot.dendrogram (plot) or plot (plotSilhouettes).

k, h	Desired number of clusters or tree height to be used for cutting the dendrogram, respectively. One or the other must be specified. Analogous to cutree.
maxTreeHeight, deepSplit, minModuleSize	Arguments used by cutreeDynamicTree.
x, obj	A componentsClust (derived) object.
pal	Colour palette to be used from **RColorBrewer**.
numericLabels	Set to TRUE to label with numeric indices instead of (long) feature group names.
colourBranches	Whether branches from cut clusters (and their labels) should be coloured. Might be slow with large numbers of clusters, hence, the default is only TRUE when this is not the case.
showLegend	If TRUE and colourBranches is also TRUE then a legend will be shown which outlines cluster numbers and their colours. By default TRUE for small amount of clusters to avoid overflowing the plot.
kSeq	An integer vector containing the sequence that should be used for average silhouette width calculation.
pch, type	Passed to plot.

Methods (by generic)

- clusters(componentsClust): Accessor method to the clust slot, which was generated by hclust.
- cutClusters(componentsClust): Accessor method to the cutClusters slot. Returns a vector with cluster membership for each candidate (format as cutree).
- clusterProperties(componentsClust): Returns a list with properties on how the clustering was performed.
- treeCut(componentsClust): Manually (re-)cut the dendrogram.
- treeCutDynamic(componentsClust): Automatically (re-)cut the dendrogram using the cutreeDynamicTree function from **dynamicTreeCut**.
- plot(x = componentsClust, y = missing): generates a dendrogram from a given cluster object and optionally highlights resulting branches when the cluster is cut.
- plotSilhouettes(componentsClust): Plots the average silhouette width when the clusters are cut by a sequence of k numbers. The k value with the highest value (marked in the plot) may be considered as the optimal number of clusters.

Slots

distm Distance matrix that was used for clustering (obtained with daisy).

clust Object returned by hclust.

cutClusters A list with assigned clusters (same format as what cutree returns).

gInfo The groupInfo of the feature groups object that was used.

properties A list containing general properties and parameters used for clustering.

altered Set to TRUE if the object was altered (*e.g.* filtered) after its creation.

S4 class hierarchy

- components
  - componentsClust
    - componentsIntClust
    - componentsSpecClust

Note

The intensity values for components (used by plotSpectrum) are set to a dummy value (1) as no single intensity value exists for this kind of components.

When the object is altered (*e.g.* by filtering or subsetting it), methods that need the original clustered data such as plotting methods do not work anymore and stop with an error.

References

Schollee JE, Bourgin M, von Gunten U, McArdell CS, Hollender J (2018). "Non-target screening to trace ozonation transformation products in a wastewater treatment train including different post-treatments." *Water Research*, **142**, 267–278. doi:10.1016/j.watres.2018.05.045.

See Also

components and generateComponents

---

componentsNT-class *Components class for homologous series.*

Description

This class is derived from components and is used to store results from unsupervised homolog detection with the **nontarget** package.

Usage

```
## S4 method for signature 'componentsNT'
plotGraph(obj, onlyLinked = TRUE, width = NULL, height = NULL)

## S4 method for signature 'componentsNTSet'
plotGraph(obj, onlyLinked = TRUE, set, ...)

## S4 method for signature 'componentsNTSet'
unset(obj, set)
```

Arguments

obj	The componentsNT object to plot.
onlyLinked	If TRUE then only components with links are plotted.
width, height	Passed to visNetwork.
set	(**sets workflow**) The name of the set.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.

Details

Objects from this class are generated by generateComponentsNontarget

Value

plotGraph returns the result of visNetwork.

Methods (by generic)

- plotGraph(componentsNT): Plots an interactive network graph for linked homologous series (*i.e.* series with (partial) overlap which could not be merged). The resulting graph can be browsed interactively and allows quick inspection of series which may be related. The graph is constructed with the **igraph** package and rendered with **visNetwork**.

Slots

homol A list with homol objects for each replicate group as returned by homol.search

Sets workflows

The componentsNTSet class is applicable for sets workflows. This class is derived from componentsNT and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class workflowStepSet.
- unset Converts the object data for a specified set into a 'non-set' object (componentsNTUnset), which allows it to be used in 'regular' workflows. Only the components in the specified set are kept. Furthermore, the component names are restored to non-set specific names (see generateComponents for more details).

The following methods are changed or with new functionality:

- plotGraph Currently can only create graph networks from one set (specified by the set argument).

Note that the componentsNTSet class does not have a homol slot. Instead, the setObjects method can be used to access this data for a specific set.

References

Loos M, Singer H (2017). "Nontargeted homologue series extraction from hyphenated high resolution mass spectrometry data." *Journal of Cheminformatics*, **9**(1). doi:10.1186/s133210170197z.

Loos M, Gerber C, Corona F, Hollender J, Singer H (2015). "Accelerated Isotope Fine Structure Calculation Using Pruned Transition Trees." *Analytical Chemistry*, **87**(11), 5738-5744. https://pubs.acs.org/doi/abs/10.1021/acs.analchem.5b00941.

Antonov M, Csárdi G, Horvát S, Müller K, Nepusz T, Noom D, Salmon M, Traag V, Welles BF, Zanini F (2023). "igraph enables fast and robust network analysis across programming languages." *arXiv preprint arXiv:2311.10260*. doi:10.48550/arXiv.2311.10260.

Csárdi G, Nepusz T (2006). "The igraph software package for complex network research." *InterJournal*, **Complex Systems**, 1695. https://igraph.org.

Csárdi G, Nepusz T, Traag V, Horvát S, Zanini F, Noom D, Müller K, Schoch D, Salmon M (2025). *igraph: Network Analysis and Visualization in R*. doi:10.5281/zenodo.7682609, R package version 2.2.1, https://CRAN.R-project.org/package=igraph.

See Also

components and generateComponents

---

componentsSpecClust-class

Components based on MS/MS similarity.

Description

This class is derived from componentsClust and is used to store components from feature groups that were clustered based on their MS/MS similarities.

Details

Objects from this class are generated by generateComponentsSpecClust

S4 class hierarchy

- componentsClust
  - componentsSpecClust

Note

When the object is altered (*e.g.* by filtering or subsetting it), methods that need the original clustered data such as plotting methods do not work anymore and stop with an error.

See Also

componentsClust for other relevant methods and generateComponents

componentsTPs-class *Components based on parent and transformation product (TP) linkage.*

Description

This class is derived from components and is used to store components that result from linking feature groups that are (predicted to be) parents with feature groups that (are predicted to be) transformation products. For more details, see generateComponentsTPs.

Usage

```
## S4 method for signature 'componentsTPs'
as.data.table(x)

## S4 method for signature 'componentsTPs'
filter(
  obj,
  ...,
  retDirMatch = FALSE,
  minSpecSim = NULL,
  minSpecSimPrec = NULL,
  minSpecSimBoth = NULL,
  minFragMatches = NULL,
  minNLMatches = NULL,
  formulas = NULL,
  verbose = TRUE,
  negate = FALSE
)

## S4 method for signature 'componentsTPs'
plotGraph(obj, onlyLinked = TRUE, width = NULL, height = NULL)
```

Arguments

x, obj	A componentsTPs object.
..., verbose	Further arguments passed to the base filter method.
retDirMatch	If set to TRUE, only keep TPs for which the retention time direction (retDir, see Details in componentsTPs) matches with the observed direction. TPs will never be removed if the expected/observed direction is '0' (*i.e.* unknown or not significantly different than the parent).
minSpecSim, minSpecSimPrec, minSpecSimBoth	The minimum spectral similarity of a TP compared to its parent ('0-1'). The minSpecSimPrec and minSpecSimBoth apply to binned data that is shifted with the "precursor" and "both" method, respectively (see MS spectral similarity parameters for more details). Set to NULL to ignore.

`minFragMatches, minNLMatches`
: Minimum number of parent/TP fragment and neutral loss matches, respectively. Set to `NULL` to ignore. See the `Linking parents and transformation products` section in `generateComponentsTPs` for more details.

`formulas`
: A `formulas` object. The formula annotation data in this object is to verify if elemental additions/subtractions from metabolic logic reactions are possible (hence, it only works with data from `generateTPsLogic`). To verify elemental additions, only TPs with at least one candidate formula that has these elements are kept. Similarly, for elemental subtractions, any of the parent candidate formulae must contain the subtraction elements. Note that TPs are currently not filtered if either the parent or the TP has no formula annotations. Set to `NULL` to ignore.

`negate`
: If `TRUE` then filters are applied in opposite manner.

`onlyLinked`
: If `TRUE` then only components with links are plotted.

`width, height`
: Passed to `visNetwork`.

Value

`filter` returns a filtered `componentsTPs` object.

`plotGraph` returns the result of `visNetwork`.

Methods (by generic)

- `as.data.table(componentsTPs)`: Returns all component data as a `data.table`.
- `filter(componentsTPs)`: Provides various rule based filtering options to clean and prioritize TP data.
- `plotGraph(componentsTPs)`: Plots an interactive network graph for linked components. Components are linked with each other if one or more transformation products overlap. The graph is constructed with the **igraph** package and rendered with **visNetwork**.

Slots

`fromTPs` A `logical` that is `TRUE` when the componentization was performed with `transformationProducts` data.

S4 class hierarchy

- `components`
  - `componentsTPs`

Note

The intensity values for components (used by `plotSpectrum`) are set to a dummy value (1) as no single intensity value exists for this kind of components.

References

Antonov M, Csárdi G, Horvát S, Müller K, Nepusz T, Noom D, Salmon M, Traag V, Welles BF, Zanini F (2023). "igraph enables fast and robust network analysis across programming languages." *arXiv preprint arXiv:2311.10260*. doi:10.48550/arXiv.2311.10260.

Csárdi G, Nepusz T (2006). "The igraph software package for complex network research." *InterJournal*, **Complex Systems**, 1695. https://igraph.org.

Csárdi G, Nepusz T, Traag V, Horvát S, Zanini F, Noom D, Müller K, Schoch D, Salmon M (2025). *igraph: Network Analysis and Visualization in R*. doi:10.5281/zenodo.7682609, R package version 2.2.1, https://CRAN.R-project.org/package=igraph.

See Also

components for other relevant methods and generateComponents

---

componentTable *Component class*

Description

Contains data for feature groups that are related in some way. These *components* commonly include adducts, isotopes and homologues.

Usage

```
componentTable(obj)

componentInfo(obj)

findFGroup(obj, fGroup)

## S4 method for signature 'components'
componentTable(obj)

## S4 method for signature 'components'
componentInfo(obj)

## S4 method for signature 'components'
groupNames(obj)

## S4 method for signature 'components'
length(x)

## S4 method for signature 'components'
names(x)
```

```
## S4 method for signature 'components'
show(object)

## S4 method for signature 'components,ANY,ANY,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'components,ANY,ANY'
x[[i, j]]

## S4 method for signature 'components'
x$name

## S4 method for signature 'components'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'components'
as.data.table(x)

## S4 method for signature 'components'
filter(
  obj,
  size = NULL,
  adducts = NULL,
  isotopes = NULL,
  rtIncrement = NULL,
  mzIncrement = NULL,
  checkComponentsSession = NULL,
  negate = FALSE,
  verbose = TRUE
)

## S4 method for signature 'components'
findFGroup(obj, fGroup)

## S4 method for signature 'components'
plotSpectrum(obj, index, markFGroup = NULL, xlim = NULL, ylim = NULL, ...)

## S4 method for signature 'components'
plotChroms(obj, index, fGroups, EICParams = getDefEICParams(rtWindow = 5), ...)

## S4 method for signature 'components'
consensus(obj, ...)

## S4 method for signature 'componentsFeatures'
show(object)

## S4 method for signature 'componentsSet'
show(object)
```

```
## S4 method for signature 'componentsSet,ANY,ANY,missing'
x[i, j, ..., sets = NULL, drop = TRUE]

## S4 method for signature 'componentsSet'
filter(obj, ..., negate = FALSE, sets = NULL)

## S4 method for signature 'componentsSet'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'componentsSet'
consensus(obj, ...)

## S4 method for signature 'componentsSet'
unset(obj, set)
```

Arguments

obj, object, x	The component object.
fGroup	The name (thus a character) of the feature group that should be searched for.
i, j	For [/[[: A numeric or character value which is used to select components/feature groups by their index or name, respectively (for the order/names see names()/groupNames()). For [: Can also be logical to perform logical selection (similar to regular vectors). If missing all components/feature groups are selected. For [[: should be a scalar value. j is optional. For delete: The data to remove from. i are the components as numeric index, logical or character, j the feature groups as numeric index/logical (relative to component) or character. If either is NULL then data for all is removed. j may also be a function: it will be called for each component, with the component (a data.table), the component name and any other arguments passed as ... to delete. The return value of this function specifies the feature groups to be removed (same format as j).
...	For delete: passed to the function specified as j. For plotChroms: Further (optional) arguments passed to the plotChroms method for the featureGroups class. Note that the colourBy, showPeakArea, showFGroupRect and topMost arguments cannot be set as these are set by this method. For plotSpectrum: Further arguments passed to plot. For consensus: components objects that should be used to generate the consensus. For sets workflow methods: further arguments passed to the base components method.
drop	ignored.
name	The component name (partially matched).

size	Should be a two sized vector with the minimum/maximum size of a component. Set to NULL to ignore.
adducts	Remove any feature groups within components that do not match given adduct rules. If adducts is a logical then only results are kept when an adduct is assigned (adducts=TRUE) or not assigned (adducts=FALSE). Otherwise, if adducts contains one or more adduct objects (or something that can be converted to it with as.adduct) then only results are kept that match the given adducts. Set to NULL to ignore this filter.
isotopes	Only keep results that match a given isotope rule. If isotopes is a logical then only results are kept with (isotopes=TRUE) or without (isotopes=FALSE) isotope assignment. Otherwise isotopes should be a numeric vector with isotope identifiers to keep (*e.g.* '0' for monoisotopic results, '1' for 'M+1' results etc.). Set to NULL to ignore this filter.
rtIncrement, mzIncrement	Should be a two sized vector with the minimum/maximum retention or mz increment of a homologous series. Set to NULL to ignore.
checkComponentsSession	If set then components and/or feature groups are removed that were selected for removal (see check-GUI and the checkComponents function). The value of checkComponentsSession should either by a path to the session file or TRUE, in which case the default session file name is used. If negate=TRUE then all non-selected data is removed instead.
negate	If TRUE then filters are applied in opposite manner.
verbose	If set to FALSE then no text output is shown.
index	The index of the component. Can be a numeric index or a character with its name.
markFGroup	If specified (*i.e.* not NULL) this argument can be used to mark a feature group in the plotted spectrum. The value should be a character with the name of the feature group. Setting this to NULL will not mark any peak.
xlim, ylim	Sets the plot size limits used by plot. Set to NULL for automatic plot sizing.
fGroups	The featureGroups object that was used to generate the components.
EICParams	A named list with parameters used for extracted ion chromatogram (EIC) creation. See the EIC parameters documentation for more details.
sets	(**sets workflow**) A character with name(s) of the sets to keep (or remove if negate=TRUE).
set	(**sets workflow**) The name of the set.

Details

components objects are obtained from generateComponents.

Value

delete returns the object for which the specified data was removed.

consensus returns a components object that is produced by merging multiple specified components objects.

Methods (by generic)

- `componentTable(components)`: Accessor method for the `components` slot of a `components` class. Each component is stored as a `data.table`.
- `componentInfo(components)`: Accessor method for the `componentInfo` slot of a `components` class.
- `groupNames(components)`: returns a `character` vector with the names of the feature groups for which data is present in this object.
- `length(components)`: Obtain total number of components.
- `names(components)`: Obtain the names of all components.
- `show(components)`: Show summary information for this object.
- `x[i`: Subset on components/feature groups.
- `x[[i`: Extracts a component table, optionally filtered by a feature group.
- `$`: Extracts a component table by component name.
- `delete(components)`: Completely deletes specified (parts of) components.
- `as.data.table(components)`: Returns all component data in a table.
- `filter(components)`: Provides rule based filtering for components.
- `findFGroup(components)`: Returns the component id(s) to which a feature group belongs.
- `plotSpectrum(components)`: Plot a *pseudo* mass spectrum for a single component.
- `plotChroms(components)`: Plot an extracted ion chromatogram (EIC) for all feature groups within a single component.
- `consensus(components)`: Generates a consensus from multiple `components` objects. At this point results are simply combined and no attempt is made to merge similar components.

Slots

`components` List of all components in this object. Use the `componentTable` method for access.

`componentInfo` A `data.table` containing general information for each component. Use the `componentInfo` method for access.

S4 class hierarchy

- `workflowStep`
  - `components`
    - `componentsCamera`
    - `componentsFeatures`
      - `componentsCliqueMS`
      - `componentsOpenMS`
    - `componentsClust`
      - `componentsIntClust`
      - `componentsSpecClust`
    - `componentsSet`
      - `componentsNTSet`

    * componentsUnset
    * componentsNT
      · componentsNTUnset
    * componentsRC
    * componentsTPs

Sets workflows

The componentsSet class is applicable for sets workflows. This class is derived from components and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class workflowStepSet.
- unset Converts the object data for a specified set into a 'non-set' object (componentsUnset), which allows it to be used in 'regular' workflows. Only the components in the specified set are kept.

The following methods are changed or with new functionality:

- filter and the subset operator ([) Can be used to select components that are only present for selected sets.

Note

filter Applies only those filters for which a component has data available. For instance, filtering by adduct will only filter any results within a component if that component contains adduct information.

For plotChroms: The topMost and topMostByRGroup EIC parameters are ignored unless the components are from homologous series.

See Also

generateComponents

---

compoundsCluster-class

Compounds cluster class

Description

Objects from this class are used to store hierarchical clustering data of candidate structures within compounds objects.

Usage

```
## S4 method for signature 'compoundsCluster'
clusters(obj)

## S4 method for signature 'compoundsCluster'
cutClusters(obj)

## S4 method for signature 'compoundsCluster'
clusterProperties(obj)

## S4 method for signature 'compoundsCluster'
groupNames(obj)

## S4 method for signature 'compoundsCluster'
length(x)

## S4 method for signature 'compoundsCluster'
lengths(x, use.names = TRUE)

## S4 method for signature 'compoundsCluster'
show(object)

## S4 method for signature 'compoundsCluster,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'compoundsCluster'
treeCut(obj, k = NULL, h = NULL, groupName)

## S4 method for signature 'compoundsCluster'
treeCutDynamic(obj, maxTreeHeight, deepSplit, minModuleSize, groupName)

## S4 method for signature 'compoundsCluster,missing'
plot(
  x,
  ...,
  groupName,
  pal = "Paired",
  colourBranches = lengths(x)[groupName] < 50,
  showLegend = lengths(x)[groupName] < 20
)

## S4 method for signature 'compoundsCluster'
getMCS(obj, groupName, cluster)

## S4 method for signature 'compoundsCluster'
plotStructure(
  obj,
  groupName,
```

```
  cluster,
  width = 500,
  height = 500,
  withTitle = TRUE
)

## S4 method for signature 'compoundsCluster'
plotSilhouettes(obj, kSeq, groupName, pch = 16, type = "b", ...)
```

Arguments

obj, x, object	A compoundsCluster object.
use.names	A logical value specifying whether the returned vector should be named with the feature group names.
i	For [: A numeric or character value which is used to select feature groups by their index or name, respectively (for the order/names see groupNames()). Can also be logical to perform logical selection (similar to regular vectors). If missing all feature groups are selected.
...	Further arguments passed directly to the plotting function (plot or plot.dendrogram).
drop, j	ignored.
k, h	Desired number of clusters or tree height to be used for cutting the dendrogram, respecitively. One or the other must be specified. Analogous to cutree.
groupName	A character specifying the feature group name.
maxTreeHeight, deepSplit, minModuleSize	Arguments used by cutreeDynamicTree.
pal	Colour palette to be used from **RColorBrewer**.
colourBranches	Whether branches from cut clusters (and their labels) should be coloured. Might be slow with large numbers of clusters, hence, the default is only TRUE when this is not the case.
showLegend	If TRUE and colourBranches is also TRUE then a legend will be shown which outlines cluster numbers and their colours. By default TRUE for small amount of clusters to avoid overflowing the plot.
cluster	A numeric value specifying the cluster.
width, height	The dimensions (in pixels) of the raster image that should be plotted.
withTitle	A logical value specifying whether a title should be added.
kSeq	An integer vector containing the sequence that should be used for average silhouette width calculation.
pch, type	Passed to plot.

Details

Objects from this type are returned by the compounds method for makeHCluster.

Value

`cutTree` and `cutTreeDynamic` return the modified `compoundsCluster` object.

`getMCS` returns an **rcdk** molecule object (`IAtomContainer`).

Methods (by generic)

- `clusters(compoundsCluster)`: Accessor method to the `clusters` slot. Returns a list that contains for each feature group an object as returned by `hclust`.
- `cutClusters(compoundsCluster)`: Accessor method to the `cutClusters` slot. Returns a list that contains for each feature group a vector with cluster membership for each candidate (format as `cutree`).
- `clusterProperties(compoundsCluster)`: Returns a list with properties on how the clustering was performed.
- `groupNames(compoundsCluster)`: returns a `character` vector with the names of the feature groups for which data is present in this object.
- `length(compoundsCluster)`: Returns the total number of clusters.
- `lengths(compoundsCluster)`: Returns a `vector` with the number of clusters per feature group.
- `show(compoundsCluster)`: Show summary information for this object.
- `x[i`: Subset on feature groups.
- `treeCut(compoundsCluster)`: Manually (re-)cut a dendrogram that was generated for a feature group.
- `treeCutDynamic(compoundsCluster)`: Automatically (re-)cut a dendrogram that was generated for a feature group using the `cutreeDynamicTree` function from **dynamicTreeCut**.
- `plot(x = compoundsCluster, y = missing)`: Plot the dendrogram for clustered compounds of a feature group. Clusters are highlighted using **dendextend**.
- `getMCS(compoundsCluster)`: Calculates the maximum common substructure (MCS) for all candidate structures within a specified cluster. This method uses the `get.mcs` function from **rcdk**.
- `plotStructure(compoundsCluster)`: Plots the maximum common substructure (MCS) for all candidate structures within a specified cluster.
- `plotSilhouettes(compoundsCluster)`: Plots the average silhouette width when the clusters are cut by a sequence of k numbers. The k value with the highest value (marked in the plot) may be considered as the optimal number of clusters.

Slots

`clusters` A `list` with `hclust` objects for each feature group.

`dists` A `list` with distance matrices for each feature group.

`SMILES` A `list` containing a vector with `SMILES` for all candidate structures per feature group.

`cutClusters` A `list` with assigned clusters for all candidates per feature group (same format as what `cutree` returns).

`properties` A list containing general properties and parameters used for clustering.

compoundScorings *Scorings terms for compound candidates*

Description

Returns an overview of scorings may be applied to rank candidate compounds.

Usage

```
compoundScorings(
  algorithm = NULL,
  database = NULL,
  includeSuspectLists = TRUE,
  onlyDefault = FALSE,
  includeNoDB = TRUE
)
```

Arguments

algorithm	The algorithm: "metfrag" or "sirius". Set to NULL to return all scorings.
database	The database for which results should be returned (*e.g.* "pubchem"). Set to NULL to return all scorings.
includeSuspectLists, onlyDefault, includeNoDB	A logical specifying whether scoring terms related to suspect lists, default scoring terms and non-database specific scoring terms should be included in the output, respectively.

Value

A data.frame with information on which scoring terms are used, what their algorithm specific name is and other information such as to which database they apply and short remarks.

See Also

generateCompounds

compoundsSIRIUS-class *Compounds class for SIRIUS results.*

Description

This class is derived from compounds and contains additional specific SIRIUS data.

Details

Objects from this class are generated by generateCompoundsSIRIUS

Slots

`fingerprints` A `list` with for each feature group result a `data.table` containing fingerprints obtained with `CSI:FingerID`.

S4 class hierarchy

- `compounds`
  - `compoundsSIRIUS`

References

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**(4), 299–302. doi:10.1038/s4159201903448.

Duhrkop K, Bocker S (2015). "Fragmentation Trees Reloaded." In Przytycka TM (ed.), *Research in Computational Molecular Biology*, 65–79. ISBN 978-3-319-16706-0.

Duhrkop K, Shen H, Meusel M, Rousu J, Bocker S (2015). "Searching molecular structure databases with tandem mass spectra using CSI:FingerID." *Proceedings of the National Academy of Sciences*, **112**(41), 12580–12585. doi:10.1073/pnas.1509788112.

Bocker S, Letzel MC, Liptak Z, Pervukhin A (2008). "SIRIUS: decomposing isotope patterns for metabolite identification." *Bioinformatics*, **25**(2), 218–224. doi:10.1093/bioinformatics/btn603.

See Also

`compounds` and `generateCompoundsSIRIUS`

`convertMSFiles` *MS data conversion*

Description

Conversion of MS analysis files between several open and closed data formats.

Usage

```
MSFileFormats(algorithm = "pwiz", vendor = FALSE)

convertMSFiles(
  files = NULL,
  outPath = NULL,
  dirs = TRUE,
  anaInfo = NULL,
  from = NULL,
  to = "mzML",
```

```
  overWrite = FALSE,
  algorithm = "pwiz",
  centroid = algorithm != "openms",
  filters = NULL,
  extraOpts = NULL,
  PWizBatchSize = 1
)
```

Arguments

algorithm	Either "pwiz" (implemented by msConvert of ProteoWizard), "openms" (implemented by FileConverter of OpenMS) or "bruker" (implemented by DataAnalysis).
vendor	If TRUE only vendor formats are returned.
files, dirs	The files argument should be a character vector with input files. If files contains directories and dirs=TRUE then files from these directories are also considered. An alternative method to specify input files is by the anaInfo argument. If the latter is specified files may be NULL.
outPath	A character vector specifying directories that should be used for the output. Will be re-cycled if necessary. If NULL, output directories will be kept the same as the input directories.
anaInfo	An analysis info table used to retrieve input files. Either this argument or files (or both) should be set (*i.e.* not NULL).
from	Input format (see below). These are used to find analyses when dirs=TRUE or anaInfo is set.
to	Output format: "mzXML" or "mzML".
overWrite	Should existing destination file be overwritten (TRUE) or not (FALSE)?
centroid	Set to TRUE to enable centroiding (not supported if algorithm="openms"). In addition, when algorithm="pwiz" the value may be "vendor" to perform centroiding with the vendor algorithm or "cwt" to use ProteoWizard's wavelet algorithm.
filters	When algorithm="pwiz": a character vector specifying one or more filters. The elements of the specified vector are directly passed to the --filter option (see here)
extraOpts	A character vector specifying any extra commandline parameters passed to msConvert or FileConverter. Set to NULL to ignore. For options: see FileConverter and msConvert.
PWizBatchSize	When algorithm="pwiz": the number of analyses to process by a single call to msConvert. Usually a value of one is most efficient. Set to zero to run all analyses all at once from a single call.

Details

MSFileFormats returns a character with all supported input formats (see below).

convertMSFiles converts the data format of an analysis to another. It uses tools from ProteoWizard (msConvert command), OpenMS (FileConverter command) or Bruker DataAnalysis to perform

the conversion. Supported input and output formats include 'mzXML', '.mzML' and several vendor formats, depending on which algorithm is used.

Parallelization

convertMSFiles (except if algorithm="bruker") uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Conversion formats

Possible output formats (to argument) are mzXML and mzML.

Possible input formats (from argument) depend on the algorithm that was chosen and may include:

- thermo: Thermo '.RAW' files (only algorithm="pwiz").
- bruker: Bruker '.d', '.yep', '.baf' and '.fid' files (only algorithm="pwiz" or algorithm="bruker").
- agilent: Agilent '.d' files (only algorithm="pwiz").
- ab: AB Sciex '.wiff' files (only algorithm="pwiz").
- waters Waters '.RAW' files (only algorithm="pwiz").
- mzXML/mzML: Open format '.mzXML'/'.mzML' files (only algorithm="pwiz" or algorithm="openms").

Note that the actual supported file formats of ProteoWizard depend on how it was installed (see here).

References

Rost HL, Sachsenberg T, Aiche S, Bielow C, Weisser H, Aicheler F, Andreotti S, Ehrlich H, Gutenbrunner P, Kenar E, Liang X, Nahnsen S, Nilse L, Pfeuffer J, Rosenberger G, Rurik M, Schmitt U, Veit J, Walzer M, Wojnar D, Wolski WE, Schilling O, Choudhary JS, Malmstrom L, Aebersold R, Reinert K, Kohlbacher O (2016). "OpenMS: a flexible open-source software platform for mass spectrometry data analysis." *Nature Methods*, **13**(9), 741–748. doi:10.1038/nmeth.3959.

Chambers MC, Maclean B, Burke R, Amodei D, Ruderman DL, Neumann S, Gatto L, Fischer B, Pratt B, Egertson J, Hoff K, Kessner D, Tasman N, Shulman N, Frewen B, Baker TA, Brusniak M, Paulse C, Creasy D, Flashner L, Kani K, Moulding C, Seymour SL, Nuwaysir LM, Lefebvre B, Kuhlmann F, Roark J, Rainer P, Detlev S, Hemenway T, Huhmer A, Langridge J, Connolly B, Chadick T, Holly K, Eckels J, Deutsch EW, Moritz RL, Katz JE, Agus DB, MacCoss M, Tabb DL, Mallick P (2012). "A cross-platform toolkit for mass spectrometry and proteomics." *Nature Biotechnology*, **30**(10), 918–920. doi:10.1038/nbt.2377.

Examples

```
## Not run:
# Use FileConverter of OpenMS to convert between open mzXML/mzML format
convertMSFiles("standard-1.mzXML", to = "mzML", algorithm = "openms")

# Convert all Thermo .RAW files in the analyses/raw directory to mzML and
# store the files in analyses/mzml. During conversion files are centroided by
```

```
# the peakPicking filter and only MS 1 data is kept.
convertMSFiles("analyses/raw", "analyses/mzml", dirs = TRUE, from = "thermo",
               centroid = "vendor", filters = "msLevel 1")

## End(Not run)
```

`defaultOpenMSAdducts` *Default adducts for OpenMS componentization*

Description

Returns the default adducts and their probabilities when the OpenMS algorithm is used for componentization.

Usage

```
defaultOpenMSAdducts(ionization)
```

Arguments

`ionization`	The ionization polarity: either `"positive"` or `"negative"`.

Details

See the `potentialAdducts` argument of `generateComponentsOpenMS` for more details.

`EICParams` *Extracted Ion Chromatogram parameters*

Description

Parameters for creation of extracted ion chromatograms.

Usage

```
getDefEICParams(...)
```

Arguments

`...`	optional named arguments that override defaults.

Details

To configure the creation of extracted ion chromatograms (EICs) several parameters exist:

- `rtWindow` Retention time (in seconds) that will be subtracted/added to respectively the minimum and maximum retention time of the feature. Thus, setting this value to '`>0`' will 'zoom out' on the retention time axis.
- `topMost` Only create EICs for this number of top most intense features. If `NULL` then EICs are created for all features.
- `topMostByRGroup` If set to `TRUE` and `topMost` is set: only create EICs for the top most features in each replicate group. For instance, when `topMost=1` and `topMostByRGroup=TRUE`, then EICs will be plotted for the most intense feature of each replicate group.
- `onlyPresent` If `TRUE` then EICs are created only for analyses in which a feature was detected. If `onlyPresent=FALSE` then EICs are generated for **all** analyses. The latter is handy to evaluate if a peak was 'missed' during feature detection or removed during *e.g.* filtering.

if `onlyPresent=FALSE` then the following parameters are also relevant:

- `mzExpWindow` To create EICs for analyses in which no feature was found, the *m/z* value is derived from the min/max values of all features in the feature group. The value of `mzExpWindow` further expands this window.
- `setsAdductPos`,`setsAdductNeg` (**sets workflow**) In sets workflows the adduct must be known to calculate the ionized *m/z*. If a feature is completely absent in a particular set then it follows no adduct annotations are available and the value of `setsAdductPos` (positive ionization data) or `setsAdductNeg` (negative ionization data) will be used instead.

These parameters are passed as a named `list` as the `EICParams` argument to functions that use EICs. The `getDefEICParams` function can be used to generate such parameter list with defaults.

`feature-optimization` *Optimization of feature finding and grouping parameters*

Description

Automatic optimization of feature finding and grouping parameters through Design of Experiments (DoE).

Usage

```
optimizeFeatureGrouping(
  features,
  algorithm,
  ...,
  templateParams = list(),
  paramRanges = list(),
  maxIterations = 50,
  maxModelDeviation = 0.1,
```

```
  parallel = TRUE
)

generateFGroupsOptPSet(algorithm, ...)

getDefFGroupsOptParamRanges(algorithm)

optimizeFeatureFinding(
  anaInfo,
  algorithm,
  ...,
  templateParams = list(),
  paramRanges = list(),
  isoIdent = if (algorithm == "openms") "OpenMS" else "IPO",
  checkPeakShape = "none",
  CAMERAOpts = list(),
  maxIterations = 50,
  maxModelDeviation = 0.1,
  parallel = TRUE
)

generateFeatureOptPSet(algorithm, ...)

getDefFeaturesOptParamRanges(algorithm, method = "centWave")
```

Arguments

features	A features object with the features that should be used to optimize grouping.
algorithm	The algorithm used for finding or grouping features (see findFeatures and groupFeatures).
...	One or more lists with parameter sets (see below) (for optimizeFeatureFinding and optimizeFeatureGrouping). Alternatively, named arguments that set (and possibly override) the parameters that should be returned from generateFeatureOptPSet or generateFGroupsOptPSet.
templateParams	Template parameter set (see below).
paramRanges	A list with vectors containing absolute parameter ranges (minimum/maximum) that constrain numeric parameters choosen during experiments. See the getDefFeaturesOptParamRange and getDefFGroupsOptParamRanges functions for defaults. Values should be Inf when no limit should be used.
maxIterations	Maximum number of iterations that may be performed to find optimimum values. Used to restrict neededless long optimization procedures. In IPO this was fixed to '50'.
maxModelDeviation	See the Potential suboptimal results by optimization model section below.
parallel	If set to TRUE then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.

anaInfo	Analysis info table (passed to findFeatures).
isoIdent	Sets the algorithm used to identify isotopes. Valid values are: "IPO", "CAMERA" and "OpenMS". The latter can only be used when OpenMS is used to find features, and is highly recommended in this situation.
checkPeakShape	Additional peak shape checking of isotopes. Only used if isoIdent="IPO". Valid values: "none", "borderIntensity", "sinusCurve" or "normalDistr".
CAMERAOpts	A list with additional arguments passed to CAMERA::findIsotopes when isoIdent="CAMERA".
method	Method used by XCMS to find features (only if algorithm="xcms").

Details

Many different parameters exist that may affect the output quality of feature finding and grouping. To avoid time consuming manual experimentation, functionality is provided to largely automate the optimization process. The methodology, which uses design of experiments (DoE), is based on the excellent Isotopologue Parameter Optimization (IPO) R package. The functionality of this package is directly integrated in patRoon. Some functionality was added or changed, however, the principle algorithm workings are nearly identical.

Compared to IPO, the following functionality was added or changed:

- The code was made more generic in order to include support for other feature finding/grouping algorithms (*e.g.* OpenMS, enviPick, XCMS3).
- The methodology of FeatureFinderMetabo (OpenMS) may be used to find isotopes.
- The maxModelDeviation parameter was added to potentially avoid suboptimal results (issue discussed here).
- The use of multiple 'parameter sets' (discussed below) which, for instance, allow optimizing qualitative paremeters more easily (see examples).
- More consistent optimization code for feature finding/grouping.
- More consistent output using S4 classes (*i.e.* optimizationResult class).
- Parallelization is performed via the **future** package instead of **BiocParallel**. If this is enabled (parallel=TRUE) then any parallelization supported by the feature finding or grouping algorithm is disabled.

Value

The optimizeFeatureFinding and optimizeFeatureGrouping return their results in a optimizationResult object.

Parameter sets

Which parameters should be optimized is determined by a *parameter set*. A set is defined by a named list containing the minimum and maximum starting range for each parameter that should be tested. For instance, the set list(chromFWHM = c(5, 10), mzPPM = c(5, 15)) specifies that the chromFWHM and mzPPM parameters (used by OpenMS feature finding) should be optimized within a range of '5'-'10' and '5'-'15', respectively. Note that this range may be increased or decreased after a DoE iteration in order to find a better optimum. The absolute limits are controlled by the paramRanges function argument.

Multiple parameter sets may be specified (*i.e.* through the ... function argument). In this situation, the optimization algorithm is repeated for each set, and the final optimum is determined from the parameter set with the best response. The `templateParams` function argument may be useful in this case to define a template for each parameter set. Actual parameter sets are then constructed by joining each parameter set with the set specified for `templateParams`. When a parameter is defined in both a regular and template set, the parameter in the regular set takes precedence.

Parameters that should not be optimized but still need to be set for the feature finding/grouping functions should also be defined in a (template) parameter set. Which parameters should be optimized is determined whether its value is specified as a vector range or a single fixed value. For instance, when a set is defined as `list(chromFWHM = c(5, 10), mzPPM = 5)`, only the `chromFWHM` parameter is optimized, whereas `mzPPM` is kept constant at '5'.

Using multiple parameter sets with differing fixed values allows optimization of qualitative values (see examples below).

The parameters specified in parameter sets are directly passed through the `findFeatures` or `groupFeatures` functions. Hence, grouping and retention time alignment parameters used by XCMS should (still) be set through the `groupArgs` and `retcorArgs` parameters.

NOTE: For XCMS3, which normally uses parameter classes for settings its options, the parameters must be defined in a named list like any other algorithm. The set parameters are then used passed to the constructor of the right parameter class object (e.g. `CentWaveParam`, `ObiwarpParam`). For grouping/alignment sets, these parameters need to be specified in nested lists called `groupParams` and `retAlignParams`, respectively (similar to `groupArgs`/`retcorArgs` for `algorithm="xcms"`). Finally, the underlying XCMS method to be used should be defined in the parameter set (*i.e.* by setting the `method` field for feature parameter sets and the `groupMethod` and `retAlignMethod` for grouping/aligning parameter sets). See the examples below for more details.

NOTE: Similar to IPO, the `peakwidth` and `prefilter` parameters for XCMS feature finding should be split in two different values:

- The minimum and maximum ranges for `peakwidth` are optimized by setting `min_peakwidth` and `max_peakwidth`, respectively.
- The `k` and `I` parameters contained in `prefilter` are split in `prefilter` and `value_of_prefilter`, respectively.

Similary, for KPIC2, the following parameters should be split:

- the `width` parameter (feature optimization) is optimized by specifying the `min_width` and `max_width` parameters.
- the `tolerance` and `weight` parameters (feature grouping optimization) are optimized by setting `mz_tolerance`/`rt_tolerance` and `mz_weight`/`rt_weight` parameters, respectively.

Functions

The `optimizeFeatureFinding` and `optimizeFeatureGrouping` are the functions to be used to optimize parameters for feature finding and grouping, respectively. These functions are analogous to `optimizeXcmsSet` and `optimizeRetGroup` from **IPO**.

The `generateFeatureOptPSet` and `generateFGroupsOptPSet` functions may be used to generate a parameter set for feature finding and grouping, respectively. Some algorithm dependent default parameter optimization ranges will be returned. These functions are analogous to `getDefaultXcmsSetStartingParams`

and getDefaultRetGroupStartingParams from **IPO**. However, unlike their IPO counterparts, these functions will not output default fixed values. The generateFGroupsOptPSet will only generate defaults for density grouping if algorithm="xcms".

The getDefFeaturesOptParamRanges and getDefFGroupsOptParamRanges return the default absolute optimization parameter ranges for feature finding and grouping, respectively. These functions are useful if you want to set the paramRanges function argument.

Potential suboptimal results by optimization model

After each experiment iteration an optimimum parameter set is found by generating a model containing the tested parameters and their responses. Sometimes the actual response from the parameters derived from the model is actually signficantly lower than expected. When the response is lower than the maximum reponse found during the experiment, the parameters belonging to this experimental maximum may be choosen instead. The maxModelDeviation argument sets the maximum deviation in response between the modelled and experimental maxima. The value is relative: '0' means that experimental values will always be favored when leading to improved responses, whereas 1 will effectively disable this procedure (and return to 'regular' IPO behaviour).

Source

The code and methodology is a direct adaptation from the IPO R package.

References

Libiseller G, Dvorzak M, Kleb U, Gander E, Eisenberg T, Madeo F, Neumann S, Trausinger G, Sinner F, Pieber T, Magnes C (2015). "IPO: a tool for automated optimization of XCMS parameters." *BMC Bioinformatics*, **16**(1). doi:10.1186/s1285901505628.

Examples

```
# example data from patRoonData package
dataDir <- patRoonData::exampleDataPath()
anaInfo <- generateAnalysisInfo(dataDir)
anaInfo <- anaInfo[1:2, ] # only focus on first two analyses (e.g. training set)

# optimize mzPPM and chromFWHM parameters
ftOpt <- optimizeFeatureFinding(anaInfo, "openms", list(mzPPM = c(5, 10), chromFWHM = c(4, 8)))

# optimize chromFWHM and isotopeFilteringModel (a qualitative parameter)
ftOpt2 <- optimizeFeatureFinding(anaInfo, "openms",
                                 list(isotopeFilteringModel = "metabolites (5% RMS)"),
                                 list(isotopeFilteringModel = "metabolites (2% RMS)"),
                                 templateParams = list(chromFWHM = c(4, 8)))

# perform grouping optimization with optimized features object
fgOpt <- optimizeFeatureGrouping(optimizedObject(ftOpt), "xcms",
                                 list(groupArgs = list(bw = c(22, 28)),
                                      retcorArgs = list(method = "obiwarp")))

# same, but using the XCMS3 interface
fgOpt2 <- optimizeFeatureGrouping(optimizedObject(ftOpt), "xcms3",
```

```
                          list(groupMethod = "density", groupParams = list(bw = c(22, 28)),
                                       retAlignMethod = "obiwarp"))


# plot contour of first parameter set/DoE iteration
plot(ftOpt, paramSet = 1, DoEIteration = 1, type = "contour")

# generate parameter set with some predefined and custom parameters to be
# optimized.
pSet <- generateFeatureOptPSet("openms", chromSNR = c(3, 9),
                                useSmoothedInts = FALSE)
```

`feature-plotting` *Plotting of grouped features*

Description

Various plotting functions for feature group data.

Usage

```
## S4 method for signature 'featureGroups,missing'
plot(
  x,
  colourBy = c("none", "rGroups", "fGroups"),
  onlyUnique = FALSE,
  retMin = FALSE,
  showLegend = TRUE,
  col = NULL,
  pch = NULL,
  ...
)

## S4 method for signature 'featureGroups'
plotInt(
  obj,
  average = FALSE,
  normalized = FALSE,
  xnames = TRUE,
  showLegend = FALSE,
  pch = 20,
  type = "b",
  lty = 3,
  col = NULL,
  plotArgs = NULL,
  linesArgs = NULL
)
```

```
## S4 method for signature 'featureGroupsSet'
plotInt(
  obj,
  average = FALSE,
  normalized = FALSE,
  xnames = !sets,
  showLegend = sets,
  pch = 20,
  type = "b",
  lty = 3,
  col = NULL,
  plotArgs = NULL,
  linesArgs = NULL,
  sets = FALSE
)

## S4 method for signature 'featureGroups'
plotChord(
  obj,
  addSelfLinks = FALSE,
  addRetMzPlots = TRUE,
  average = FALSE,
  outerGroups = NULL,
  addIntraOuterGroupLinks = FALSE,
  ...
)

## S4 method for signature 'featureGroups'
plotChroms(
  obj,
  analysis = analyses(obj),
  groupName = names(obj),
  retMin = FALSE,
  showPeakArea = FALSE,
  showFGroupRect = TRUE,
  title = NULL,
  colourBy = c("none", "rGroups", "fGroups"),
  showLegend = TRUE,
  annotate = c("none", "ret", "mz"),
  intMax = "eic",
  EICParams = getDefEICParams(),
  showProgress = FALSE,
  xlim = NULL,
  ylim = NULL,
  EICs = NULL,
  ...
)
```

```
## S4 method for signature 'featureGroups'
plotVenn(obj, which = NULL, ...)

## S4 method for signature 'featureGroupsSet'
plotVenn(obj, which = NULL, ..., sets = FALSE)

## S4 method for signature 'featureGroups'
plotUpSet(obj, which = NULL, nsets = length(which), nintersects = NA, ...)

## S4 method for signature 'featureGroups'
plotVolcano(
  obj,
  FCParams,
  showLegend = TRUE,
  averageFunc = mean,
  normalized = FALSE,
  col = NULL,
  pch = 19,
  ...
)

## S4 method for signature 'featureGroups'
plotGraph(obj, onlyPresent = TRUE, width = NULL, height = NULL)

## S4 method for signature 'featureGroupsSet'
plotGraph(obj, onlyPresent = TRUE, set, ...)
```

Arguments

colourBy	Sets the automatic colour selection: "none" for a single colour or "rGroups"/"fGroups" for a distinct colour per replicate/feature group.
onlyUnique	If TRUE and colourBy="rGroups" then only feature groups that are unique to a replicate group are plotted.
retMin	Plot retention time in minutes (instead of seconds).
showLegend	Plot a legend if TRUE.
col	Colour(s) used. If col=NULL then colours are automatically generated.
pch, type, lty	Common plotting parameters passed to *e.g.* plot. For plot: if pch=NULL then values are automatically assigned.
...	passed to plot (plot, plotChroms, plotTICs and plotBPCs), **VennDiagram** plotting functions (plotVenn), chordDiagram (plotChord) or upset (plotUpSet).
obj, x	featureGroups object to be used for plotting.
average	If TRUE then data within replicate groups are averaged. For as.data.table: if features=TRUE other feature properties are also averaged.
xnames	Plot analysis (or replicate group if average=TRUE) names on the x axis.

plotArgs, linesArgs
: A list with further arguments passed to plot and lines, respectively.

sets
: (**sets workflow**) For plotInt: if TRUE then feature intensities are plot per set (order follows the analysis information).

  For plotVenn: If TRUE then the which argument changes its meaning and is used to specify the names of the sets to be compared.

addSelfLinks
: If TRUE then 'self-links' are added which represent non-shared data.

addRetMzPlots
: Set to TRUE to enable *m/z vs* retention time scatter plots.

outerGroups
: Character vector of names to be used as outer groups. The values in the specified vector should be named by analysis names (average set to FALSE) or replicate group names (average set to TRUE), for instance: c(analysis1 = "group1", analysis2 = "group1", analysis3 = "group2"). Set to NULL to disable outer groups.

addIntraOuterGroupLinks
: If TRUE then links will be added within outer groups.

analysis, groupName
: character vector with the analyses/group names to be considered for plotting. Compared to subsetting the featureGroups object (obj) upfront this is slightly faster and (if onlyPresent=FALSE) allows plotting chromatograms for feature groups where none of the specified analyses contain the feature (which is impossible otherwise since subsetting leads to removal of 'empty' feature groups).

showPeakArea
: Set to TRUE to display integrated chromatographic peak ranges by filling (shading) their areas.

showFGroupRect
: Set to TRUE to mark the full retention/intensity range of all features within a feature group by drawing a rectangle around it.

title
: Character string used for title of the plot. If NULL a title will be automatically generated.

annotate
: If set to "ret" and/or "mz" then retention and/or *m/z* values will be drawn for each plotted feature group.

intMax
: Method used to determine the maximum intensity plot limit. Should be "eic" (from EIC data) or "feature" (from feature data). Ignored if the ylim parameter is specified.

EICParams
: A named list with parameters used for extracted ion chromatogram (EIC) creation. See the EIC parameters documentation for more details.

showProgress
: if set to TRUE then a text progressbar will be displayed when all EICs are being plot. Set to "none" to disable any annotation.

xlim, ylim
: Sets the plot size limits used by plot. Set to NULL for automatic plot sizing.

EICs
: Internal parameter for now and should be kept at NULL (default).

which
: A character vector with replicate groups used for comparison. Set to NULL to ignore.

  For plotVenn: alternatively a named list containing elements of character vectors with replicate groups to compare. For instance, which=list(infl = c("influent-A", "influent-B"), effl = c("effluent-A", "effluent-B")), will compare the features in replicate groups '"influent-A/B"' against those

in '"effluent-A/B"'. The names of the list are used for labelling in the plot, and will be made automatically if not specified.

`nsets`, `nintersects`
: See `upset`.

`FCParams`
: A parameter list to calculate Fold change data. See `getFCParams` for more details.

`averageFunc`, `normalized`
: Used for intensity data treatment, see the documentation for the `as.data.table` method.

`onlyPresent`
: Only plot feature groups of internal standards that are still present in the `featureGroups` input object (which may be otherwise be removed by *e.g.* subsetting or `filter`).

`width`, `height`
: Passed to `visNetwork`.

`set`
: (**sets workflow**) The set for which data must be plotted.

Details

`plot` Generates an *m/z vs* retention time plot for all featue groups. Optionally highlights unique/overlapping presence amongst replicate groups.

`plotInt` Generates a line plot for the (averaged) intensity of feature groups within all analyses

`plotChord` Generates a chord diagram which can be used to visualize shared presence of feature groups between analyses or replicate groups. In addition, analyses/replicates sharing similar properties (*e.g.* location, age, type) may be grouped to enhance visualization between these 'outer groups'.

`plotChroms` Plots extracted ion chromatograms (EICs) of feature groups.

`plotVenn` plots a Venn diagram (using **VennDiagram**) outlining unique and shared feature groups between up to five replicate groups.

`plotUpSet` plots an UpSet diagram (using the `upset` function) outlining unique and shared feature groups between given replicate groups.

`plotVolcano` Plots Fold change data in a 'Volcano plot'.

`plotGraph` generates an interactive network plot which is used to explore internal standard (IS) assignments to each feature group. This requires the availability of IS assignments, see the documentation for `normInts` for details. The graph is rendered with **visNetwork**.

Value

`plotVenn` (invisibly) returns a list with the following fields:

- `gList` the `gList` object that was returned by the utilized **VennDiagram** plotting function.
- `areas` The total area for each plotted group.
- `intersectionCounts` The number of intersections between groups.

The order for the `areas` and `intersectionCounts` fields is the same as the parameter order from the used plotting function (see *e.g.* `draw.pairwise.venn` and `draw.triple.venn`).

`plotGraph` returns the result of `visNetwork`.

Sets workflows

The following methods are changed or with new functionality:

- `plotVenn` and `plotInt` allow to handle data per set. See the `sets` argument description.
- `plotGraph` only plots data per set, and requires the `set` argument to be set.

Author(s)

Rick Helmus <<r.helmus@uva.nl>> and Ricardo Cunha <<cunha@iuta.de>> (`plotTICs` and `plotBPCs` functions)

References

Gu Z, Gu L, Eils R, Schlesner M, Brors B (2014). "circlize implements and enhances circular visualization in R." *Bioinformatics*, **30**, 2811-2812.

Conway JR, Lex A, Gehlenborg N (2017). "UpSetR: an R package for the visualization of intersecting sets and their properties." *Bioinformatics*, **33**(18), 2938-2940. doi:10.1093/bioinformatics/btx364, http://dx.doi.org/10.1093/bioinformatics/btx364.

Lex A, Gehlenborg N, Strobelt H, Vuillemot R, Pfister H (2014). "UpSet: Visualization of Intersecting Sets." *IEEE Transactions on Visualization and Computer Graphics*, **20**(12), 1983–1992. doi:10.1109/tvcg.2014.2346248.

See Also

`featureGroups-class`, `groupFeatures`

`featureAnnotations-class` *Base feature annotations class*

Description

Holds information for all feature group annotations.

Usage

```
## S4 method for signature 'featureAnnotations'
annotations(obj)

## S4 method for signature 'featureAnnotations'
groupNames(obj)

## S4 method for signature 'featureAnnotations'
length(x)

## S4 method for signature 'featureAnnotations,ANY,missing,missing'
```

```
x[i, j, ..., drop = TRUE]

## S4 method for signature 'featureAnnotations,ANY,missing'
x[[i, j]]

## S4 method for signature 'featureAnnotations'
x$name

## S4 method for signature 'featureAnnotations'
as.data.table(
  x,
  fGroups = NULL,
  fragments = FALSE,
  countElements = NULL,
  countFragElements = NULL,
  OM = FALSE,
  normalizeScores = "none",
  excludeNormScores = defaultExclNormScores(x)
)

## S4 method for signature 'featureAnnotations'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featureAnnotations'
filter(
  obj,
  minExplainedPeaks = NULL,
  scoreLimits = NULL,
  elements = NULL,
  fragElements = NULL,
  lossElements = NULL,
  topMost = NULL,
  OM = FALSE,
  negate = FALSE
)

## S4 method for signature 'featureAnnotations'
plotVenn(obj, ..., labels = NULL, vennArgs = NULL)

## S4 method for signature 'featureAnnotations'
plotUpSet(
  obj,
  ...,
  labels = NULL,
  nsets = length(list(...)) + 1,
  nintersects = NA,
  upsetArgs = NULL
)
```

Arguments

obj, x	featureAnnotations object to be accessed
i, j	For [/[[: A numeric or character value which is used to select feature groups by their index or name, respectively (for the order/names see groupNames()). For [: Can also be logical to perform logical selection (similar to regular vectors). If missing all feature groups are selected. For [[: should be a scalar value. For delete: The data to remove from. i are the feature groups as numeric index, logical or character, j the candidates as numeric indices (rows). If either is NULL then data for all is removed. j may also be a function: it will be called for each feature group, with the annotation table (a data.table), the feature group name and any other arguments passed as ... to delete. The return value of this function specifies the candidate indices (rows) to be removed (specified as an integer or logical vector).
...	For the "[" operator: ignored. For delete: passed to the function specified as j. Others: Any further (and unique) featureAnnotations objects.
drop	ignored.
name	The feature group name (partially matched).
fGroups	The featureGroups object that was used to generate this object. If not NULL it is used to add feature group information (retention and *m/z* values).
fragments	If TRUE then information on annotated fragments will be included. Automatically set to TRUE if countFragElements is set.
countElements, countFragElements	A character vector with elements that should be counted for each candidate's formula. For instance, c("C", "H") adds columns for both carbon and hydrogen amounts of each formula. Note that the neutral formula (neutral_formula column) is used to count elements of non-fragmented formulae, whereas the charged formula of fragments (ion_formula column in fragInfo data) is used for fragments. Set to NULL to not count any elements.
OM	For as.data.table: if set to TRUE several columns with information relevant for organic matter (OM) characterization will be added (e.g. elemental ratios, classification). This will also make sure that countElements contains at least C, H, N, O, P and S. For filter: If TRUE then several filters are applied to exclude unlikely formula candidates present in organic matter (OM). See Source section for details.
normalizeScores	A character that specifies how normalization of annotation scorings occurs. Either "none" (no normalization), "max" (normalize to max value) or "minmax" (perform min-max normalization). Note that normalization of negative scores (e.g. output by SIRIUS) is always performed as min-max. Furthermore, currently normalization for compounds takes the original min/max scoring values

into account when candidates were generated. Thus, for compounds scoring, normalization is not affected when candidate results were removed after they were generated (*e.g.* by use of filter).

excludeNormScores
: A character vector specifying any compound scoring names that should *not* be normalized. Set to NULL to normalize all scorings. Note that whether any normalization occurs is set by the excludeNormScores argument.

  For compounds: By default score and individualMoNAScore are set to mimic the behavior of the MetFrag web interface.

minExplainedPeaks
: Minimum number of explained peaks. Set to NULL to ignore.

scoreLimits
: Filter results by their scores. Should be a named list that contains two-sized numeric vectors with the minimum/maximum value of a score (use -Inf/Inf for no limits). The names of each element should follow the name column of the table returned by formulaScorings$name and compoundScorings()$name. For instance, scoreLimits=list(numberPatents=c(10, Inf)) specifies that numberPatents should be at least '10'. Note that a result without a specified scoring is never removed. If a score term exists multiple times, *i.e.* due to a consensus, then a candidate is kept if at least one of the terms falls within the range. Set to NULL to skip this filter.

elements
: Only retain candidate formulae (neutral form) that match a given elemental restriction. The format of elements is a character string with elements that should be present where each element is followed by a valid amount or a range thereof. If no number is specified then '1' is assumed. For instance, elements="C1-10H2-20O0-2P", specifies that '1-10', '2-20', '0-2' and '1' carbon, hydrogen, oxygen and phosphorus atoms should be present, respectively. When length(elements)>1 formulas are tested to follow at least one of the given elemental restrictions. For instance, elements=c("P", "S") specifies that either one phosphorus or one sulfur atom should be present. Set to NULL to ignore this filter.

fragElements, lossElements
: Specifies elemental restrictions for fragment or neutral loss formulae (charged form). Candidates are retained if at least one of the fragment formulae follow (or not follow if negate=TRUE) the given restrictions. See elements for the used format.

topMost
: Only keep a maximum of topMost candidates with highest score (or least highest if negate=TRUE). Set to NULL to ignore.

negate
: If TRUE then filters are applied in opposite manner.

labels
: A character with names to use for labelling. If NULL labels are automatically generated.

vennArgs
: A list with further arguments passed to **VennDiagram** plotting functions. Set to NULL to ignore.

nsets, nintersects
: See upset.

upsetArgs
: A list with any further arguments to be passed to upset. Set to NULL to ignore.

Details

This class stores annotation data for feature groups, such as molecular formulae, SMILES identifiers, compound names etc. The class of objects that are generated by formula and compound annotation (`generateFormulas` and `generateCompounds`) are based on this class.

Value

`as.data.table` returns a `data.table`.

`delete` returns the object for which the specified data was removed.

`filter` returns a filtered `featureAnnotations` object.

`plotVenn` (invisibly) returns a list with the following fields:

- `gList` the `gList` object that was returned by the utilized **VennDiagram** plotting function.
- `areas` The total area for each plotted group.
- `intersectionCounts` The number of intersections between groups.

The order for the `areas` and `intersectionCounts` fields is the same as the parameter order from the used plotting function (see *e.g.* `draw.pairwise.venn` and `draw.triple.venn`).

Methods (by generic)

- `annotations(featureAnnotations)`: Accessor for the `groupAnnotations` slot.
- `groupNames(featureAnnotations)`: returns a `character` vector with the names of the feature groups for which data is present in this object.
- `length(featureAnnotations)`: Obtain total number of candidates.
- `x[i`: Subset on feature groups.
- `x[[i`: Extracts annotation data for a feature group.
- `$`: Extracts annotation data for a feature group.
- `as.data.table(featureAnnotations)`: Generates a table with all annotation data for each feature group and other information such as element counts.
- `delete(featureAnnotations)`: Completely deletes specified annotations.
- `filter(featureAnnotations)`: Provides rule based filtering for feature group annotations. Useful to eliminate unlikely candidates and speed up further processing.
- `plotVenn(featureAnnotations)`: plots a Venn diagram (using **VennDiagram**) outlining unique and shared candidates of up to five different `featureAnnotations` objects.
- `plotUpSet(featureAnnotations)`: plots an UpSet diagram (using the `upset` function) outlining unique and shared candidates between different `featureAnnotations` objects.

Slots

`groupAnnotations` A `list` with for each annotated feature group a `data.table` with annotation data. Use the `annotations` method for access.

`scoreTypes` A `character` with all the score types present in this object.

`scoreRanges` The minimum and maximum score values of all candidates for each feature group. Used for normalization.

Source

Calculation of the aromaticity index (AI) and related double bond equivalents (DBE_AI) is performed as described in Koch 2015. Formula classification is performed by the rules described in Abdulla 2013. Filtering of OM related molecules is performed as described in Koch 2006 and Kujawinski 2006. (see references).

S4 class hierarchy

- workflowStep
  - featureAnnotations
    - formulas
      - formulasConsensus
      - formulasSet
      - formulasUnset
      - formulasSIRIUS
    - compounds
      - compoundsConsensus
      - compoundsMF
      - compoundsSet
      - compoundsUnset
      - compoundsSIRIUS

References

Koch BP, Dittmar T (2015). "From mass to structure: an aromaticity index for high-resolution mass data of natural organic matter." *Rapid Communications in Mass Spectrometry*, **30**(1), 250–250. doi:10.1002/rcm.7433.

Abdulla HA, Sleighter RL, Hatcher PG (2013). "Two Dimensional Correlation Analysis of Fourier Transform Ion Cyclotron Resonance Mass Spectra of Dissolved Organic Matter: A New Graphical Analysis of Trends." *Analytical Chemistry*, **85**(8), 3895–3902. doi:10.1021/ac303221j.

Koch BP, Dittmar T (2006). "From mass to structure: an aromaticity index for high-resolution mass data of natural organic matter." *Rapid Communications in Mass Spectrometry*, **20**(5), 926–932. doi:10.1002/rcm.2386.

Kujawinski EB, Behn MD (2006). "Automated Analysis of Electrospray Ionization Fourier Transform Ion Cyclotron Resonance Mass Spectra of Natural Organic Matter." *Analytical Chemistry*, **78**(13), 4363–4373. doi:10.1021/ac0600306.

Conway JR, Lex A, Gehlenborg N (2017). "UpSetR: an R package for the visualization of intersecting sets and their properties." *Bioinformatics*, **33**(18), 2938-2940. doi:10.1093/bioinformatics/btx364, http://dx.doi.org/10.1093/bioinformatics/btx364.

Lex A, Gehlenborg N, Strobelt H, Vuillemot R, Pfister H (2014). "UpSet: Visualization of Intersecting Sets." *IEEE Transactions on Visualization and Computer Graphics*, **20**(12), 1983–1992. doi:10.1109/tvcg.2014.2346248.

See Also

formulas-class and compounds-class

The derived formulas and compounds classes.

`featureGroupsComparison-class`

Feature groups comparison class

Description

This class is used for comparing different featureGroups objects.

Usage

```
## S4 method for signature 'featureGroupsComparison'
names(x)

## S4 method for signature 'featureGroupsComparison'
length(x)

## S4 method for signature 'featureGroupsComparison,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'featureGroupsComparison,ANY,missing'
x[[i, j]]

## S4 method for signature 'featureGroupsComparison'
x$name
```

Arguments

x	A featureGroupsComparison object.
i	For [/[[: A numeric or character value which is used to select labels by their index or name, respectively (for the order/names see names()). For [: Can also be logical to perform logical selection (similar to regular vectors). If missing all labels are selected. For [[: should be a scalar value.
...	Ignored.
drop, j	ignored.
name	The label name (partially matched).

Details

Objects from this class are returned by comparison.

Methods (by generic)

- names(featureGroupsComparison): Obtain the labels that were given to each compared feature group.
- length(featureGroupsComparison): Number of feature groups objects that were compared.
- x[i: Subset on labels that were assigned to compared feature groups.
- x[[i: Extract a featureGroups object by its label.
- $: Extract a compound table for a feature group.

Slots

fGroupsList A list of featureGroups object that were compared

comparedFGroups A *pseudo* featureGroups object containing grouped feature groups.

---

featureQualityNames *Returns chromatographic peak quality and score names for features and/or feature groups.*

Description

Returns chromatographic peak quality and score names for features and/or feature groups.

Usage

```
featureQualityNames(feat = TRUE, group = TRUE, scores = FALSE, totScore = TRUE)
```

Arguments

feat	If TRUE then names specific to features are returned.
group	If TRUE then names specific to groups are returned.
scores	If TRUE the score names are returned, otherwise the quality names.
totScore	If TRUE (and scores=TRUE) then the name of the total score is included.

features-class *Base features class*

Description

Holds information for all features present within a set of analysis.

Usage

```
## S4 method for signature 'features'
length(x)

## S4 method for signature 'features'
show(object)

## S4 method for signature 'features'
featureTable(obj)

## S4 method for signature 'features'
analysisInfo(obj)

## S4 method for signature 'features'
analyses(obj)

## S4 method for signature 'features'
replicateGroups(obj)

## S4 method for signature 'features'
as.data.table(x)

## S4 method for signature 'features'
filter(
  obj,
  absMinIntensity = NULL,
  relMinIntensity = NULL,
  retentionRange = NULL,
  mzRange = NULL,
  mzDefectRange = NULL,
  chromWidthRange = NULL,
  qualityRange = NULL,
  negate = FALSE
)

## S4 method for signature 'features,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'features,ANY,missing'
```

```
x[[i]]

## S4 method for signature 'features'
x$name

## S4 method for signature 'features'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'features'
calculatePeakQualities(obj, weights, flatnessFactor, parallel = TRUE)

## S4 method for signature 'features'
getTICs(obj, retentionRange = NULL, MSLevel = 1)

## S4 method for signature 'features'
getBPCs(obj, retentionRange = NULL, MSLevel = 1)

## S4 method for signature 'features'
plotTICs(
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  colourBy = c("none", "analyses", "rGroups"),
  showLegend = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'features'
plotBPCs(
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  colourBy = c("none", "analyses", "rGroups"),
  showLegend = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'featuresSet'
sets(obj)
```

```
## S4 method for signature 'featuresSet'
show(object)

## S4 method for signature 'featuresSet'
as.data.table(x)

## S4 method for signature 'featuresSet,ANY,missing,missing'
x[i, ..., sets = NULL, drop = TRUE]

## S4 method for signature 'featuresSet'
filter(obj, ..., negate = FALSE, sets = NULL)

## S4 method for signature 'featuresSet'
unset(obj, set)

## S4 method for signature 'featuresKPIC2'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featuresXCMS'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featuresXCMS3'
delete(obj, i = NULL, j = NULL, ...)
```

Arguments

obj, x, object	features object to be accessed
absMinIntensity, relMinIntensity	Minimum absolute/relative intensity for features to be kept. The relative intensity is determined from the feature with highest intensity (within the same analysis). Set to '0' or NULL to skip this step.
retentionRange, mzRange, mzDefectRange, chromWidthRange	Range of retention time (in seconds), *m/z*, mass defect (defined as the decimal part of *m/z* values) or chromatographic peak width (in seconds), respectively. Features outside this range will be removed. Should be a numeric vector with length of two containing the min/max values. The maximum can be Inf to specify no maximum range. Set to NULL to skip this step.
qualityRange	Used to filter features by their peak qualities/scores (see calculatePeakQualities). Should be a named list with min/max ranges for each quality/score to be filtered (the featureQualityNames function can be used to obtain valid names). Example: qualityRange=list(ModalityScore=c(0.3, Inf),SymmetryScore=c(0.5, Inf)). Set to NULL to ignore.
negate	If set to TRUE then filtering operations are performed in opposite manner.
i, j	For [/[[: A numeric or character value which is used to select analyses by their index or name, respectively (for the order/names see analyses()). For [: Can also be logical to perform logical selection (similar to regular vectors). If missing all analyses are selected.

	For [[: should be a scalar value. For delete: The data to remove from. i are the analyses as numeric index, logical or character, j the features as numeric index (row) of the feature. If either is NULL then data for all is removed. j may also be a function: it will be called for each analysis, with the feature table (a data.table), the analysis name and any other arguments passed as ... to delete. The return value of this function specifies the feature indices (rows) to be removed (specified as an integer or logical vector).
...	For delete: passed to the function specified as j. For plotTICs and plotBPCs: further arguments passed to plot. For sets workflow methods: further arguments passed to the base features method.
drop	ignored.
name	The analysis name (partially matched).
weights	A named numeric vector that defines the weight for each score to calculate the totalScore. The names of the vector follow the score names. Unspecified weights are defaulted to '1'. Example: weights=c(ApexBoundaryRatioScore=0.5, GaussianSimilarityScore=2).
flatnessFactor	Passed to **MetaClean** as the flatness.factor argument to calculateJaggedness and calculateModality.
parallel	If set to TRUE then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.
MSLevel	Integer vector with the ms levels (i.e., 1 for MS1 and 2 for MS2) to obtain TIC traces.
retMin	Plot retention time in minutes (instead of seconds).
title	Character string used for title of the plot. If NULL a title will be automatically generated.
colourBy	Sets the automatic colour selection: "none" for a single colour or "analyses"/"rGroups" for a distinct colour per analysis or analysis replicate group.
showLegend	Plot a legend if TRUE.
xlim, ylim	Sets the plot size limits used by plot. Set to NULL for automatic plot sizing.
sets	(**sets workflow**) For [and filter: a character with name(s) of the sets to keep (or remove if negate=TRUE).
set	(**sets workflow**) The name of the set.

Details

This class provides a way to store intensity, retention times, *m/z* and other data for all features in a set of analyses. The class is virtual and derived objects are created by 'feature finders' such as findFeaturesOpenMS, findFeaturesXCMS and findFeaturesBruker.

Value

`featureTable`: A `list` containing a `data.table` for each analysis with feature data

`analysisInfo`: A `data.frame` containing a column with analysis name (`analysis`), its path (`path`), and other columns such as replicate group name (`group`) and blank reference (`blank`).

`delete` returns the object for which the specified data was removed.

`calculatePeakQualities` returns a modified object amended with peak qualities and scores.

Methods (by generic)

- `length(features)`: Obtain total number of features.
- `show(features)`: Shows summary information for this object.
- `featureTable(features)`: Get table with feature information
- `analysisInfo(features)`: Get analysis information
- `analyses(features)`: returns a `character` vector with the names of the analyses for which data is present in this object.
- `replicateGroups(features)`: returns a `character` vector with the names of the replicate groups for which data is present in this object.
- `as.data.table(features)`: Returns all feature data in a table.
- `filter(features)`: Performs common rule based filtering of features. Note that this (and much more) functionality is also provided by the `filter` method defined for `featureGroups`. However, filtering a `features` object may be useful to avoid grouping large amounts of features.
- `x[i`: Subset on analyses.
- `x[[i`: Extract a feature table for an analysis.
- `$`: Extract a feature table for an analysis.
- `delete(features)`: Completely deletes specified features.
- `calculatePeakQualities(features)`: Calculates peak qualities for each feature. This uses MetaClean R package to calculate the following metrics: `Apex-Boundary Ratio`, `FWHM2Base`, `Jaggedness`, `Modality`, `Symmetry`, `Gaussian Similarity`, `Sharpness`, `Triangle Peak Area Similarity Ratio` and `Zig-Zag index`. Please see the **MetaClean** publication (referenced below) for more details. For each metric, an additional score is calculated by normalizing all feature values (unless the quality metric definition has a fixed range) and scale from '`0`' (worst) to '`1`' (best). Then, a `totalScore` for each feature is calculated by the (weighted) sum of all score values.
- `getTICs(features)`: Obtain the total ion chromatogram/s (TICs) of the analyses.
- `getBPCs(features)`: Obtain the base peak chromatogram/s (BPCs) of the analyses.
- `plotTICs(features)`: Plots the TICs of the analyses.
- `plotBPCs(features)`: Plots the BPCs of the analyses.

Slots

`features` List of features per analysis file. Use the `featureTable` method for access.

`analysisInfo` Analysis group information. Use the `analysisInfo` method for access.

S4 class hierarchy

- workflowStep
  - features
    - featuresSet
    - featuresUnset
    - featuresFromFeatGroups
    - featuresConsensus
    - featuresBruker
    - featuresEnviPick
    - featuresKPIC2
    - featuresOpenMS
    - featuresSAFD
    - featuresSIRIUS
    - featuresBrukerTASQ
    - featuresXCMS
    - featuresXCMS3

Sets workflows

The `featuresSet` class is applicable for sets workflows. This class is derived from `features` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- `sets` Returns the set names for this object.
- `unset` Converts the object data for a specified set into a 'non-set' object (`featuresUnset`), which allows it to be used in 'regular' workflows. The adduct annotations for the selected set (*e.g.* as passed to `makeSet`) are used to convert all feature masses to ionic *m/z* values.

The following methods are changed or with new functionality:

- `filter` and the subset operator (`[`) have specific arguments to choose/filter by (feature presence in) sets. See the `sets` argument description.

Note

For `calculatePeakQualities`: sometimes **MetaClean** may return `NA` for the `Gaussian Similarity` metric, in which case it will be set to '`0`'.

Author(s)

Rick Helmus <<r.helmus@uva.nl>> and Ricardo Cunha <<cunha@iuta.de>> (`getTICs`, `getBPCs`, `plotTICs` and `plotBPCs` functions)

References

Chetnik K, Petrick L, Pandey G (2020). "MetaClean: a machine learning-based classifier for reduced false positive peak detection in untargeted LC-MS metabolomics data." *Metabolomics*, **16**(11). doi:10.1007/s11306020017383.

See Also

findFeatures

---

findFeatures *Finding features*

Description

Automatically find features.

Usage

```
findFeatures(analysisInfo, algorithm, ..., verbose = TRUE)
```

Arguments

analysisInfo	A data.frame with Analysis information.
algorithm	A character string describing the algorithm that should be used: "bruker", "openms", "xcms", "xcms3", "envipick", "sirius", "kpic2", "safd"
...	Further parameters passed to the selected feature finding algorithms.
verbose	If set to FALSE then no text output is shown.

Details

Several functions exist to collect features (*i.e.* retention and MS information that represent potential compounds) from a set of analyses. All 'feature finders' return an object derived from the features base class. The next step in a general workflow is to group and align these features across analyses with groupFeatures. Note that some feature finders have a plethora of options which sometimes may have a large effect on the quality of results. Fine-tuning parameters is therefore important, and the optimum is largely dependent upon applied analysis methodology and instrumentation.

findFeatures is a generic function that will find features by one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as findFeaturesOpenMS and findFeaturesXCMS. While these functions may be called directly, findFeatures provides a generic interface and is therefore usually preferred.

Value

An object of a class which is derived from features.

Note

In most cases it will be necessary to centroid your MS input files. The only exception is Bruker, however, you will still need centroided 'mzXML'/'mzML' files for *e.g.* plotting chromatograms. In this case the centroided MS files should be stored in the same directory as the raw Bruker '.d' files. The convertMSFiles function can be used to centroid data.

See Also

The features output class and its methods and the algorithm specific functions: findFeaturesBruker, findFeaturesOpenMS, findFeaturesXCMS, findFeaturesXCMS3, findFeaturesEnviPick, findFeaturesSIRIUS, findFeaturesKPIC2, findFeaturesSAFD

---

findFeaturesBruker *Find features using Bruker DataAnalysis*

Description

Uses the 'Find Molecular Features' (FMF) algorithm of Bruker DataAnalysis vendor software to find features.

Usage

```
findFeaturesBruker(
  analysisInfo,
  doFMF = "auto",
  startRange = 0,
  endRange = 0,
  save = TRUE,
  close = save,
  verbose = TRUE
)
```

Arguments

analysisInfo	A data.frame with Analysis information.
doFMF	Run the 'Find Molecular Features' algorithm before loading compounds. Valid options are: "auto" (run FMF automatically if current results indicate it is necessary) and "force" (run FMF *always*, even if cached results exist). Note that checks done if doFMF="auto" are fairly simplistic, hence set doFMF="force" if feature data needs to be updated.
startRange, endRange	Start/End retention range (seconds) from which to collect features. A 0 (zero) for endRange marks the end of the analysis.
close, save	If TRUE then Bruker files are closed and saved after processing with DataAnalysis, respectively. Setting close=TRUE prevents that many analyses might be opened simultaneously in DataAnalysis, which otherwise may use excessive memory or become slow. By default save is TRUE when close is TRUE, which is likely what you want as otherwise any processed data is lost.
verbose	If set to FALSE then no text output is shown.

Details

This function uses Bruker to automatically find features. This function is called when calling `findFeatures` with `algorithm="bruker"`.

The resulting 'compounds' are transferred from DataAnalysis and stored as features.

This algorithm only works with Bruker data files (`.d` extension) and requires Bruker DataAnalysis and the **RDCOMClient** package to be installed. Furthermore, DataAnalysis combines multiple related masses in a feature (*e.g.* isotopes, adducts) but does not report the actual (monoisotopic) mass of the feature. Therefore, it is simply assumed that the feature mass equals that of the highest intensity mass peak.

Value

An object of a class which is derived from `features`.

Note

If any errors related to `DCOM` appear it might be necessary to terminate DataAnalysis (note that DataAnalysis might still be running as a background process). The `ProcessCleaner` application installed with DataAnalayis can be used for this.

See Also

`findFeatures` for more details and other algorithms.

`findFeaturesEnviPick` *Find features using enviPick*

Description

Uses the `enviPickwrap` function from the **enviPick** R package to extract features.

Usage

```
findFeaturesEnviPick(analysisInfo, ..., parallel = TRUE, verbose = TRUE)
```

Arguments

`analysisInfo`	A `data.frame` with Analysis information.
`...`	Further parameters passed to `enviPickwrap`.
`parallel`	If set to `TRUE` then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.
`verbose`	If set to `FALSE` then no text output is shown.

Details

This function uses enviPick to automatically find features. This function is called when calling `findFeatures` with `algorithm="envipick"`.

The input MS data files need to be centroided. The `convertMSFiles` function can be used to centroid data.

Value

An object of a class which is derived from `features`.

Note

The analysis files must be in the `mzXML` format.

See Also

`findFeatures` for more details and other algorithms.

`findFeaturesKPIC2` *Find features using KPIC2*

Description

Uses the KPIC2 R package to extract features.

Usage

```
findFeaturesKPIC2(
  analysisInfo,
  kmeans = TRUE,
  level = 1000,
  ...,
  parallel = TRUE,
  verbose = TRUE
)
```

Arguments

`analysisInfo`	A `data.frame` with Analysis information.
`kmeans`	If `TRUE` then `getPIC.kmeans` is used to obtain PICs, otherwise it is `getPIC`.
`level`	Passed to `getPIC` or `getPIC.kmeans`
`...`	Further parameters passed to `getPIC`/`getPIC.kmeans`
`parallel`	If set to `TRUE` then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.
`verbose`	If set to `FALSE` then no text output is shown.

Details

This function uses KPIC2 to automatically find features. This function is called when calling `findFeatures` with `algorithm="kpic2"`.

The MS files should be in the `mzML` or `mzXML` format.

The input MS data files need to be centroided. The `convertMSFiles` function can be used to centroid data.

Value

An object of a class which is derived from `features`.

References

Ji H, Zeng F, Xu Y, Lu H, Zhang Z (2017). "KPIC2: An Effective Framework for Mass Spectrometry-Based Metabolomics Using Pure Ion Chromatograms." *Analytical Chemistry*, **89**(14), 7631–7640. doi:10.1021/acs.analchem.7b01547.

See Also

`findFeatures` for more details and other algorithms.

`findFeaturesOpenMS` *Find features using OpenMS*

Description

uses the FeatureFinderMetabo TOPP tool (see `http://www.openms.de`) to find features.

Usage

```
findFeaturesOpenMS(
  analysisInfo,
  noiseThrInt = 1000,
  chromSNR = 3,
  chromFWHM = 5,
  mzPPM = 10,
  reEstimateMTSD = TRUE,
  traceTermCriterion = "sample_rate",
  traceTermOutliers = 5,
  minSampleRate = 0.5,
  minTraceLength = 3,
  maxTraceLength = -1,
  widthFiltering = "fixed",
  minFWHM = 1,
  maxFWHM = 30,
  traceSNRFiltering = FALSE,
```

```
  localRTRange = 10,
  localMZRange = 6.5,
  isotopeFilteringModel = "metabolites (5% RMS)",
  MZScoring13C = FALSE,
  useSmoothedInts = TRUE,
  extraOpts = NULL,
  intSearchRTWindow = 3,
  useFFMIntensities = FALSE,
  verbose = TRUE
)
```

Arguments

analysisInfo	A data.frame with Analysis information.
noiseThrInt	Noise intensity threshold. Sets algorithm:common:noise_threshold_int option.
chromSNR	Minimum S/N of a mass trace. Sets algorithm:common:chrom_peak_snr option.
chromFWHM	Expected chromatographic peak width (in seconds). Sets algorithm:common:chrom_fwhm option.
mzPPM	Allowed mass deviation (ppm) for trace detection. Sets algorithm:mtd:mass_error_ppm.
reEstimateMTSD	If TRUE then enables dynamic re-estimation of m/z variance during mass trace collection stage. Sets algorithm:mtd:reestimate_mt_sd.
traceTermCriterion, traceTermOutliers, minSampleRate	Termination criterion for the extension of mass traces. See FeatureFinderMetabo. Sets the algorithm:mtd:trace_termination_criterion, algorithm:mtd:trace_termination_outl and algorithm:mtd:min_sample_rate options, respectively.
minTraceLength, maxTraceLength	Minimum/Maximum length of mass trace (seconds). Set negative value for maxlength to disable maximum. Sets algorithm:mtd:min_trace_length and algorithm:mtd:min_trace_length, respectively.
widthFiltering, minFWHM, maxFWHM	Enable filtering of unlikely peak widths. See FeatureFinderMetabo. Sets algorithm:epd:width_filter algorithm:epd:min_fwhm and algorithm:epd:max_fwhm, respectively.
traceSNRFiltering	If TRUE then apply post-filtering by signal-to-noise ratio after smoothing. Sets the algorithm:epd:masstrace_snr_filtering option.
localRTRange, localMZRange	Retention/MZ range where to look for coeluting/isotopic mass traces. Sets the algorithm:ffm:local_rt_range and algorithm:ffm:local_mz_range options, respectively.
isotopeFilteringModel	Remove/score candidate assemblies based on isotope intensities. See FeatureFinderMetabo. Sets the algorithm:ffm:isotope_filtering_model option.
MZScoring13C	Use the 13C isotope as the expected shift for isotope mass traces. See FeatureFinderMetabo. Sets algorithm:ffm:mz_scoring_13C.

useSmoothedInts
: If TRUE then use LOWESS intensities instead of raw intensities. Sets the algorithm:ffm:use_smoothed_ option.

extraOpts
: Named list containing extra options that will be passed to FeatureFinderMetabo. Any options specified here will override any of the above. Example: extraOpts=list("-algorithm:com (corresponds to setting noiseThrInt=1000). Set to NULL to ignore.

intSearchRTWindow
: Retention time window (in seconds, +/- feature retention time) that is used to find the closest data point to the retention time to obtain the intensity of a feature (this is needed since OpenMS does not provide this data).

useFFMIntensities
: If TRUE then peak intensities are directly loaded from FeatureFinderMetabo output. Otherwise, intensities are loaded afterwards from the input 'mzML' files, which is potentially much slower, especially with many analyses files. However, useFFMIntensities=TRUE is still somewhat experimental, may be less accurate and requires a recent version of OpenMS (>=2.7).

verbose
: If set to FALSE then no text output is shown.

Details

This function uses OpenMS to automatically find features. This function is called when calling findFeatures with algorithm="openms".

This functionality has been tested with OpenMS version >= 2.0. Please make sure it is installed and configured, e.g. by installing patRoonExt or configuring the path of the binaries with the patRoon.path.OpenMS option or the system 'PATH' variable.

The file format of analyses must be 'mzML'.

The input MS data files need to be centroided. The convertMSFiles function can be used to centroid data.

Value

An object of a class which is derived from features.

Parallelization

findFeaturesOpenMS uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Note that for caching purposes, the analyses files must always exist on the local host computer, even if it is not participating in computations.

References

Rost HL, Sachsenberg T, Aiche S, Bielow C, Weisser H, Aicheler F, Andreotti S, Ehrlich H, Gutenbrunner P, Kenar E, Liang X, Nahnsen S, Nilse L, Pfeuffer J, Rosenberger G, Rurik M, Schmitt U, Veit J, Walzer M, Wojnar D, Wolski WE, Schilling O, Choudhary JS, Malmstrom L, Aebersold R, Reinert K, Kohlbacher O (2016). "OpenMS: a flexible open-source software platform for mass spectrometry data analysis." *Nature Methods*, **13**(9), 741–748. doi:10.1038/nmeth.3959.

pugixml (via Rcpp) is used to process OpenMS XML output.

Eddelbuettel D (2013). *Seamless R and C++ Integration with Rcpp*. Springer, New York. doi:10.1007/9781461468684, ISBN 978-1-4614-6867-7.

Eddelbuettel D, Balamuta J (2018). "Extending R with C++: A Brief Introduction to Rcpp." *The American Statistician*, **72**(1), 28-36. doi:10.1080/00031305.2017.1375990.

Eddelbuettel D, Francois R, Allaire J, Ushey K, Kou Q, Russell N, Ucar I, Bates D, Chambers J (2025). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.0, https://www.rcpp.org.

Eddelbuettel D, François R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.1

See Also

findFeatures for more details and other algorithms.

`findFeaturesSAFD` *Find features using SAFD*

Description

Uses SAFD to obtain features. This functionality is still experimental. Please see the details below.

Usage

```
findFeaturesSAFD(
  analysisInfo,
  profPath = NULL,
  mzRange = c(0, 400),
  maxNumbIter = 1000,
  maxTPeakW = 300,
  resolution = 30000,
  minMSW = 0.02,
  RThreshold = 0.75,
  minInt = 2000,
  sigIncThreshold = 5,
  S2N = 2,
  minPeakWS = 3,
  verbose = TRUE
)
```

Arguments

analysisInfo	A data.frame with Analysis information.
profPath	A character vector with paths to the profile MS data for each analysis (will be re-cycled if necessary). See the Using SAFD section for more details.
mzRange	The *m/z* window to be imported (passed to the import_files_MS1 function).
maxNumbIter, maxTPeakW, resolution, minMSW, RThreshold, minInt, sigIncThreshold, S2N, minPeakWS	Parameters directly passed to the safd_s3D function.
verbose	If set to FALSE then no text output is shown.

Details

This function uses SAFD to automatically find features. This function is called when calling findFeatures with algorithm="safd".

The support for SAFD is still experimental, and its interface might change in the future.

In order to use SAFD, please make sure that its julia packages are installed and you have verified that everything works, *e.g.* by running the test data.

This algorithm supports profile and centroided MS data. If the use of profile data is desired, centroided data must still be available for other functionality of patRoon. The centroided data is specified through the 'regular' analysis info mechanism. The location to any profile data is specified through the profPath argument (NULL for no profile data). The base file names (*i.e.* the file name without path and extension) of both centroid and profile data must be the same. Furthermore, the format of the profile data must be 'mzXML'.

Value

An object of a class which is derived from features.

Parallelization

findFeaturesSAFD uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Note that for caching purposes, the analyses files must always exist on the local host computer, even if it is not participating in computations.

References

Samanipour S, OBrien JW, Reid MJ, Thomas KV (2019). "Self Adjusting Algorithm for the Nontargeted Feature Detection of High Resolution Mass Spectrometry Coupled with Liquid Chromatography Profile Data." *Analytical Chemistry*, **91**(16), 10800–10807. doi:10.1021/acs.analchem.9b02422.

See Also

findFeatures for more details and other algorithms.

`findFeaturesSIRIUS` *Find features using SIRIUS*

Description

Uses SIRIUS to find features.

Usage

```
findFeaturesSIRIUS(analysisInfo, verbose = TRUE)
```

Arguments

`analysisInfo`	A `data.frame` with Analysis information.
`verbose`	If set to `FALSE` then no text output is shown.

Details

This function uses SIRIUS to automatically find features. This function is called when calling `findFeatures` with `algorithm="sirius"`.

The features are collected by running the `lcms-align SIRIUS` command for every analysis.

The MS files should be in the '`mzML`' or '`mzXML`' format. Furthermore, this algorithms requires the presence of (data-dependent) MS/MS data.

The input MS data files need to be centroided. The `convertMSFiles` function can be used to centroid data.

Value

An object of a class which is derived from `features`.

Parallelization

`findFeaturesSIRIUS` uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Note that for caching purposes, the analyses files must always exist on the local host computer, even if it is not participating in computations.

References

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**(4), 299–302. doi:10.1038/s4159201903448.

See Also

`findFeatures` for more details and other algorithms.

`findFeaturesXCMS` *Find features using XCMS (old interface)*

Description

Uses the legacy `xcmsSet` function from the **xcms** package to find features.

Usage

```
findFeaturesXCMS(analysisInfo, method = "centWave", ..., verbose = TRUE)
```

Arguments

`analysisInfo`	A `data.frame` with Analysis information.
`method`	The method setting used by XCMS peak finding, see `xcms::findPeaks`
`...`	Further parameters passed to `xcmsSet`.
`verbose`	If set to `FALSE` then no text output is shown.

Details

This function uses XCMS to automatically find features. This function is called when calling `findFeatures` with `algorithm="xcms"`.

This function uses the legacy interface of **xcms**. It is recommended to use `findFeaturesXCMS3` instead.

The file format of analyses must be `mzML` or `mzXML`.

The input MS data files need to be centroided. The `convertMSFiles` function can be used to centroid data.

Value

An object of a class which is derived from `features`.

References

Benton HP, Want EJ, Ebbels TMD (2010). "Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data." *BIOINFORMATICS*, **26**, 2488.

Smith, C.A., Want, E.J., O'Maille, G., Abagyan,R., Siuzdak, G. (2006). "XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification." *Analytical Chemistry*, **78**, 779–787.

Tautenhahn R, Boettcher C, Neumann S (2008). "Highly sensitive feature detection for high resolution LC/MS." *BMC Bioinformatics*, **9**, 504.

See Also

findFeatures for more details and other algorithms.

findFeaturesXCMS3

---

findFeaturesXCMS3 *Find features using XCMS (new interface)*

Description

Uses the new xcms3 interface from the **xcms** package to find features.

Usage

```
findFeaturesXCMS3(
  analysisInfo,
  param = xcms::CentWaveParam(),
  ...,
  verbose = TRUE
)
```

Arguments

analysisInfo	A data.frame with Analysis information.
param	The method parameters used by XCMS peak finding, see xcms::findChromPeaks
...	Further parameters passed to xcms::findChromPeaks.
verbose	If set to FALSE then no text output is shown.

Details

This function uses XCMS3 to automatically find features. This function is called when calling findFeatures with algorithm="xcms3".

The file format of analyses must be mzML or mzXML.

The input MS data files need to be centroided. The convertMSFiles function can be used to centroid data.

Value

An object of a class which is derived from features.

References

Benton HP, Want EJ, Ebbels TMD (2010). "Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data." *BIOINFORMATICS*, **26**, 2488.

Smith, C.A., Want, E.J., O'Maille, G., Abagyan,R., Siuzdak, G. (2006). "XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification." *Analytical Chemistry*, **78**, 779–787.

Tautenhahn R, Boettcher C, Neumann S (2008). "Highly sensitive feature detection for high resolution LC/MS." *BMC Bioinformatics*, **9**, 504.

See Also

`findFeatures` for more details and other algorithms.

---

formulas-class *Formula annotations class*

Description

Contains data of generated chemical formulae for given feature groups.

Usage

```
## S4 method for signature 'formulas'
annotations(obj, features = FALSE)

## S4 method for signature 'formulas'
analyses(obj)

## S4 method for signature 'formulas'
defaultExclNormScores(obj)

## S4 method for signature 'formulas'
show(object)

## S4 method for signature 'formulas,ANY,ANY'
x[[i, j]]

## S4 method for signature 'formulas'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'formulas'
as.data.table(
  x,
  fGroups = NULL,
  fragments = FALSE,
```

```
  countElements = NULL,
  countFragElements = NULL,
  OM = FALSE,
  normalizeScores = "none",
  excludeNormScores = defaultExclNormScores(x),
  average = FALSE
)

## S4 method for signature 'formulas'
annotatedPeakList(
  obj,
  index,
  groupName,
  analysis = NULL,
  MSPeakLists,
  onlyAnnotated = FALSE
)

## S4 method for signature 'formulas'
plotSpectrum(
  obj,
  index,
  groupName,
  analysis = NULL,
  MSPeakLists,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'formulas'
plotScores(
  obj,
  index,
  groupName,
  analysis = NULL,
  normalizeScores = "max",
  excludeNormScores = defaultExclNormScores(obj)
)

## S4 method for signature 'formulas'
consensus(
  obj,
  ...,
  absMinAbundance = NULL,
```

```
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE,
  rankWeights = 1,
  labels = NULL
)

## S4 method for signature 'formulasSet'
show(object)

## S4 method for signature 'formulasSet'
delete(obj, i, j, ...)

## S4 method for signature 'formulasSet,ANY,missing,missing'
x[i, j, ..., sets = NULL, updateConsensus = FALSE, drop = TRUE]

## S4 method for signature 'formulasSet'
filter(obj, ..., sets = NULL, updateConsensus = FALSE, negate = FALSE)

## S4 method for signature 'formulasSet'
plotSpectrum(
  obj,
  index,
  groupName,
  analysis = NULL,
  MSPeakLists,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  perSet = TRUE,
  mirror = TRUE,
  ...
)

## S4 method for signature 'formulasSet'
annotatedPeakList(obj, index, groupName, analysis = NULL, MSPeakLists, ...)

## S4 method for signature 'formulasSet'
consensus(
  obj,
  ...,
  absMinAbundance = NULL,
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE,
  rankWeights = 1,
```

```
  labels = NULL,
  filterSets = FALSE,
  setThreshold = 0,
  setThresholdAnn = 0,
  setAvgSpecificScores = FALSE
)

## S4 method for signature 'formulasSet'
unset(obj, set)

## S4 method for signature 'formulasConsensusSet'
unset(obj, set)

## S4 method for signature 'formulasSIRIUS'
delete(obj, i = NULL, j = NULL, ...)
```

Arguments

obj, x, object	The formulas object.
features	If TRUE returns formula data for features, otherwise for feature groups.
i, j	For [[: If both i and j are specified then i specifies the analysis and j the feature group of the feature for which annotations should be returned. Otherwise i specifies the feature group for which group annotations should be returned. i/j can be specified as integer index or as a character name. Otherwise passed to the featureAnnotations method.
...	For plotSpectrum: Further arguments passed to plot. For delete: passed to the function specified as j. For consensus: Any further (and unique) formulas objects. For sets workflow methods: further arguments passed to the base formulas method.
fGroups, fragments, countElements, countFragElements, OM	Passed to the featureAnnotations method.
normalizeScores	A character that specifies how normalization of annotation scorings occurs. Either "none" (no normalization), "max" (normalize to max value) or "minmax" (perform min-max normalization). Note that normalization of negative scores (e.g. output by SIRIUS) is always performed as min-max. Furthermore, currently normalization for compounds takes the original min/max scoring values into account when candidates were generated. Thus, for compounds scoring, normalization is not affected when candidate results were removed after they were generated (*e.g.* by use of filter).
excludeNormScores	A character vector specifying any compound scoring names that should *not* be normalized. Set to NULL to normalize all scorings. Note that whether any normalization occurs is set by the excludeNormScores argument. For compounds: By default score and individualMoNAScore are set to mimic the behavior of the MetFrag web interface.

`average` If set to `TRUE` an 'average formula' is generated for each feature group by combining all elements from all candidates and averaging their amounts. This obviously leads to non-existing formulae, however, this data may be useful to deal with multiple candidate formulae per feature group when performing elemental characterization. Setting this to `TRUE` disables reporting of most other data.

`index` The candidate index (row). For `plotSpectrum` two indices can be specified to compare spectra. In this case `groupName` and `analysis` (if not `NULL`) should specify values for the spectra to compare.

`groupName` The name of the feature group (or feature groups when comparing spectra) to which the candidate belongs.

`analysis` A `character` specifying the analysis (or analyses when comparing spectra) for which the annotated spectrum should be plotted. If `NULL` then annotation results for the complete feature group will be plotted.

`MSPeakLists` The `MSPeakLists` object that was used to generate the candidate

`onlyAnnotated` Set to `TRUE` to filter out any peaks that could not be annotated.

`title` The title of the plot. Set to `NULL` for an automatically generated title.

`specSimParams` A named `list` with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.

`mincex` The formula annotation labels are automatically scaled. The `mincex` argument forces a minimum `cex` value for readability.

`xlim`, `ylim` Sets the plot size limits used by `plot`. Set to `NULL` for automatic plot sizing.

`absMinAbundance`, `relMinAbundance`
Minimum absolute or relative ('`0-1`') abundance across objects for a result to be kept. For instance, `relMinAbundance=0.5` means that a result should be present in at least half of the number of compared objects. Set to '`NULL`' to ignore and keep all results. Limits cannot be set when `uniqueFrom` is not `NULL`.

`uniqueFrom` Set this argument to only retain formulas that are unique within one or more of the objects for which the consensus is made. Selection is done by setting the value of `uniqueFrom` to a `logical` (values are recycled), `numeric` (select by index) or a `character` (as obtained with `algorithm(obj)`). For `logical` and `numeric` values the order corresponds to the order of the objects given for the consensus. Set to `NULL` to ignore.

`uniqueOuter` If `uniqueFrom` is not `NULL` and if `uniqueOuter=TRUE`: only retain data that are also unique between objects specified in `uniqueFrom`.

`rankWeights` A numeric vector with weights of to calculate the mean ranking score for each candidate. The value will be re-cycled if necessary, hence, the default value of '`1`' means equal weights for all considered objects.

`labels` A `character` with names to use for labelling. If `NULL` labels are automatically generated.

`sets` (**sets workflow**) A `character` with name(s) of the sets to keep (or remove if `negate=TRUE`). Note: if `updateConsensus=FALSE` then the `setCoverage` column of the annotation results is not updated.

`updateConsensus`
(**sets workflow**) If `TRUE` then the annonation consensus among set results is updated. See the `Sets workflows` section for more details.

drop	Passed to the featureAnnotations method.
negate	Passed to the featureAnnotations method.
perSet, mirror	(**sets workflow**) If perSet=TRUE then the set specific mass peaks are annotated separately. Furthermore, if mirror=TRUE (and there are two sets in the object) then a mirror plot is generated.
filterSets	(**sets workflow**) Controls how algorithms concensus abundance filters are applied. See the Sets workflows section below.
setThreshold, setThresholdAnn	(**sets workflow**) Thresholds used to create the annotation set consensus. See generateFormulas.
setAvgSpecificScores	(**sets workflow**) If TRUE then set specific annotation scores (*e.g.* MS/MS and isotopic pattern match scores) are averaged for the set consensus. See generateFormulas.
set	(**sets workflow**) The name of the set.

Details

formulas objects are obtained with generateFormulas. This class is derived from the featureAnnotations class, please see its documentation for more methods and other details.

Value

annotations returns a list containing for each feature group (or feature if features=TRUE) a data.table with an overview of all generated formulae and other data such as candidate scoring and MS/MS fragments.

consensus returns a formulas object that is produced by merging results from multiple formulas objects.

Methods (by generic)

- annotations(formulas): Accessor method to obtain generated formulae.
- analyses(formulas): returns a character vector with the names of the analyses for which data is present in this object.
- defaultExclNormScores(formulas): Returns default scorings that are excluded from normalization.
- show(formulas): Show summary information for this object.
- x[[i: Extracts a formula table, either for a feature group or for features in an analysis.
- as.data.table(formulas): Generates a table with all candidate formulae for each feature group and other information such as element counts.
- annotatedPeakList(formulas): Returns an MS/MS peak list annotated with data from a given candidate formula.
- plotSpectrum(formulas): Plots an annotated spectrum for a given candidate formula of a feature or feature group. Two spectra can be compared by specifying a two-sized vector for the index, groupName and (if desired) analysis arguments.

- `plotScores(formulas)`: Plots a barplot with scoring of a candidate formula.
- `consensus(formulas)`: Generates a consensus of results from multiple objects. In order to rank the consensus candidates, first each of the candidates are scored based on their original ranking (the scores are normalized and the highest ranked candidate gets value '1'). The (weighted) mean is then calculated for all scorings of each candidate to derive the final ranking (if an object lacks the candidate its score will be '0'). The original rankings for each object is stored in the `rank` columns.

Slots

`featureFormulas` A `list` with all generated formulae for each analysis/feature group. Use the `annotations` method for access.

`setThreshold,setThresholdAnn,setAvgSpecificScores` (**sets workflow**) A copy of the equally named arguments that were passed when this object was created by `generateFormulas`.

`origFGNames` (**sets workflow**) The original (order of) names of the `featureGroups` object that was used to create this object.

S4 class hierarchy

- `featureAnnotations`
  - `formulas`
    - `formulasConsensus`
    - `formulasSet`
      - `formulasConsensusSet`
    - `formulasUnset`
    - `formulasSIRIUS`

Source

Subscripting of formulae for plots generated by `plotSpectrum` is based on the `chemistry2expression` function from the ReSOLUTION package.

Sets workflows

The `formulasSet` class is applicable for sets workflows. This class is derived from `formulas` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class `workflowStepSet`.
- `unset` Converts the object data for a specified set into a 'non-set' object (`formulasUnset`), which allows it to be used in 'regular' workflows. Only the annotation results that are present in the specified set are kept (based on the set consensus, see below for implications).

The following methods are changed or with new functionality:

- `filter` and the subset operator (`[`) Can be used to select data that is only present for selected sets. Depending on the `updateConsenus`, both either operate on set consensus or original data (see below for implications).

- annotatedPeakList Returns a combined annotation table with all sets.
- plotSpectrum Is able to highlight set specific mass peaks (perSet and mirror arguments).
- consensus Creates the algorithm consensus based on the original annotation data (see below for implications). Then, like the sets workflow method for generateFormulas, a consensus is made for all sets, which can be controlled with the setThreshold and setThresholdAnn arguments. The candidate coverage among the different algorithms is calculated for each set (*e.g.* coverage-positive column) and for all sets (coverage column), which is based on the presence of a candidate in all the algorithms from all sets data. The consensus method for sets workflow data supports the filterSets argument. This controls how the algorithm consensus abundance filters (absMinAbundance/relMinAbundance) are applied: if filterSets=TRUE then the minimum of all coverage set specific columns is used to obtain the algorithm abundance. Otherwise the overall coverage column is used. For instance, consider a consensus object to be generated from two objects generated by different algorithms (*e.g.* SIRIUS and GenForm), which both have a positive and negative set. Then, if a candidate occurs with both algorithms for the positive mode set, but only with the first algorithm in the negative mode set, relMinAbundance=1 will remove the candidate if filterSets=TRUE (because the minimum relative algorithm abundance is '0.5'), while filterSets=FALSE will not remove the candidate (because based on all sets data the candidate occurs in both algorithms).

Two types of annotation data are stored in a formulasSet object:

1. Annotations that are produced from a consensus between set results (see generateFormulas).
2. The 'original' annotation data per set, prior to when the set consensus was made. This includes candidates that were filtered out because of the thresholds set by setThreshold and setThresholdAnn. However, when filter or subsetting ([) operations are performed, the original data is also updated.

In most cases the first data is used. However, in a few cases the original annotation data is used (as indicated above), for instance, to re-create the set consensus. It is important to realize that the original annotation data may have *additional* candidates, and a newly created set consensus may therefore have 'new' candidates. For instance, when the object consists of the sets "positive" and "negative" and setThreshold=1 was used to create it, then formulas[, sets = "positive", updateConsensus = TRUE] may now have additional candidates, *i.e.* those that were not present in the "negative" set and were previously removed due to the consensus threshold filter.

See Also

The featureAnnotations base class for more relevant methods and generateFormulas.

---

formulaScorings *Scorings terms for formula candidates*

Description

Returns a data.frame with information on which scoring terms are used and what their algorithm specific name is.

Usage

```
formulaScorings()
```

See Also

generateFormulas

`formulasSIRIUS-class` *Formulas class for SIRIUS results.*

Description

This class is derived from `formulas` and contains additional specific `SIRIUS` data.

Details

Objects from this class are generated by `generateFormulasSIRIUS`

Slots

`fingerprints` A `list` with for each feature group result a `data.table` containing fingerprints obtained with `CSI:FingerID`. Will be empty unless the `getFingerprints` argument to `generateFormulasSIRIUS` was set to `TRUE`.

`MS2QuantMeta` Metadata from **MS2Quant** filled in by `predictRespFactors`.

S4 class hierarchy

- `formulas`
  - `formulasSIRIUS`

References

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**(4), 299–302. doi:10.1038/s4159201903448.

Duhrkop K, Bocker S (2015). "Fragmentation Trees Reloaded." In Przytycka TM (ed.), *Research in Computational Molecular Biology*, 65–79. ISBN 978-3-319-16706-0.

Duhrkop K, Shen H, Meusel M, Rousu J, Bocker S (2015). "Searching molecular structure databases with tandem mass spectra using CSI:FingerID." *Proceedings of the National Academy of Sciences*, **112**(41), 12580–12585. doi:10.1073/pnas.1509788112.

Bocker S, Letzel MC, Liptak Z, Pervukhin A (2008). "SIRIUS: decomposing isotope patterns for metabolite identification." *Bioinformatics*, **25**(2), 218–224. doi:10.1093/bioinformatics/btn603.

See Also

formulas and generateFormulasSIRIUS

---

generateComponents *Grouping feature groups in components*

Description

Functionality to automatically group related feature groups (*e.g.* isotopes, adducts and homologues) to assist and simplify annotation.

Usage

```
generateComponents(fGroups, algorithm, ...)

## S4 method for signature 'featureGroups'
generateComponents(fGroups, algorithm, ...)
```

Arguments

fGroups	featureGroups object for which components should be generated.
algorithm	A character string describing the algorithm that should be used: "ramclustr", "camera", "nontarget", "intclust", "openms", "cliquems", "specclust", "tp"
...	Any parameters to be passed to the selected component generation algorithm.

Details

Several algorithms are provided to group feature groups that are related in some (chemical) way to each other. How feature groups are related depends on the algorithm: examples include adducts, statistics and parents/transformation products. The linking of this data is generally useful for annotation purposes and reducing data complexity.

generateComponents is a generic function that will generateComponents by one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as generateComponentsRAMClustR and generateComponentsNontarget. While these functions may be called directly, generateComponents provides a generic interface and is therefore usually preferred.

Value

A components (derived) object containing all generated components.

Sets workflows

In a sets workflow the componentization data is generated differently depending on the used algorithm. Please see the details in the algorithm specific functions linked in the See Also section.

See Also

The components output class and its methods and the algorithm specific functions: generateComponentsRAMClustR, generateComponentsCAMERA, generateComponentsNontarget, generateComponentsIntClust, generateComponentsOpenMS, generateComponentsCliqueMS, generateComponentsSpecClust, generateComponentsTPs

---

generateComponentsCAMERA

Componentization of adducts, isotopes etc. with CAMERA

Description

Interfaces with CAMERA to generate components from known adducts, isotopes and in-source fragments.

Usage

```
generateComponentsCAMERA(fGroups, ...)

## S4 method for signature 'featureGroups'
generateComponentsCAMERA(
  fGroups,
  ionization = NULL,
  onlyIsotopes = FALSE,
  minSize = 2,
  relMinReplicates = 0.5,
  extraOpts = NULL
)

## S4 method for signature 'featureGroupsSet'
generateComponentsCAMERA(fGroups, ionization = NULL, ...)
```

Arguments

fGroups	featureGroups object for which components should be generated.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
ionization	Which ionization polarity was used to generate the data: should be "positive" or "negative". If the featureGroups object has adduct annotations, and ionization=NULL, the ionization will be detected automatically. (**sets workflow**) This parameter is not supported for sets workflows, as the ionization will always be detected automatically.
onlyIsotopes	Logical value. If TRUE only isotopes are considered when generating components (faster). Corresponds to quick argument of CAMERA::annotate.
minSize	The minimum size of a component. Smaller components than this size will be removed. See note below.

relMinReplicates
: Feature groups within a component are only kept when they contain data for at least this (relative) amount of replicate analyses. For instance, '0.5' means that at least half of the replicates should contain data for a particular feature group in a component. In this calculation replicates that are fully absent within a component are not taken in to account. See note below.

extraOpts
: Named character vector with extra arguments directly passed to CAMERA::annotate. Set to NULL to ignore.

Details

This function uses CAMERA to generate components. This function is called when calling generateComponents with algorithm="camera".

The specified featureGroups object is automatically converted to an xcmsSet object using getXCMSSet.

Value

A components (derived) object containing all generated components.

Sets workflows

In a sets workflow the componentization is first performed for each set independently. The resulting components are then all combined in a componentsSet object. Note that the components themselves are never merged. The components are renamed to include the set name from which they were generated (*e.g.* "CMP1" becomes "CMP1-positive").

Note

The default value for minSize and relMinReplicates results in extra filtering, hence, the final results may be different than what the algorithm normally would return.

References

Kuhl C, Tautenhahn R, Boettcher C, Larson TR, Neumann S (2012). "CAMERA: an integrated strategy for compound spectra extraction and annotation of liquid chromatography/mass spectrometry data sets." *Analytical Chemistry*, **84**, 283–289. http://pubs.acs.org/doi/abs/10.1021/ac202450g.

See Also

generateComponents for more details and other algorithms.

`generateComponentsCliqueMS` *Componentization of adducts, isotopes etc. with cliqueMS*

Description

Uses cliqueMS to generate components using the `cliqueMS::getCliques` function.

Usage

```
generateComponentsCliqueMS(fGroups, ...)

## S4 method for signature 'featureGroups'
generateComponentsCliqueMS(
  fGroups,
  ionization = NULL,
  maxCharge = 1,
  maxGrade = 2,
  ppm = 10,
  adductInfo = NULL,
  absMzDev = 0.005,
  minSize = 2,
  relMinAdductAbundance = 0.75,
  adductConflictsUsePref = TRUE,
  NMConflicts = c("preferential", "mostAbundant", "mostIntense"),
  prefAdducts = c("[M+H]+", "[M-H]-"),
  extraOptsCli = NULL,
  extraOptsIso = NULL,
  extraOptsAnn = NULL,
  parallel = TRUE
)

## S4 method for signature 'featureGroupsSet'
generateComponentsCliqueMS(fGroups, ionization = NULL, ...)
```

Arguments

`fGroups`	`featureGroups` object for which components should be generated.
`...`	(**sets workflow**) Further arguments passed to the non-sets workflow method.
`ionization`	Which ionization polarity was used to generate the data: should be `"positive"` or `"negative"`. If the `featureGroups` object has adduct annotations, and `ionization=NULL`, the ionization will be detected automatically. (**sets workflow**) This parameter is not supported for sets workflows, as the ionization will always be detected automatically.
`maxCharge, maxGrade, ppm`	Arguments passed to `cliqueMS::getIsotopes` and/or `cliqueMS::getAnnotation`.

`adductInfo` Sets the `adinfo` argument to `cliqueMS::getAnnotation`. If `NULL` then the default adduct information from **cliqueMS** is used (*i.e.* the `positive.adinfo`/`negative.adinfo` package datasets).

`absMzDev` Maximum absolute *m/z* deviation.

`minSize` The minimum size of a component. Smaller components than this size will be removed. See note below.

`relMinAdductAbundance` The minimum relative abundance ('`0-1`') that an adduct should be assigned to features within the same feature group. See the `Feature components` section for more details.

`adductConflictsUsePref` If set to `TRUE`, and not all adduct assigments to the features within a feature group are equal and at least one of those adducts is a preferential adduct (`prefAdducts` argument), then only the features with (the lowest ranked) preferential adduct are considered. In all other cases or when `adductConflictsUsePref=FALSE` only features with the most frequently assigned adduct is considered. See the `Feature components` section for more details.

`NMConflicts` The strategies to employ when not all neutral masses within a component are equal. Valid options are: `"preferential"`, `"mostAbundant"` and `"mostIntense"`. Multiple strategies are possible, and will be executed in the given order until one succeeds. See the `Feature components` section for more details.

`prefAdducts` A `character` vector with one or more *preferential adducts*. See the `Feature components` section for more details.

`extraOptsCli`, `extraOptsIso`, `extraOptsAnn` Named `list` with further arguments to be passed to `cliqueMS::getCliques`, `cliqueMS::getIsotopes` and `cliqueMS::getAnnotation`, respectively. Set to `NULL` to ignore.

`parallel` If set to `TRUE` then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.

Details

This function uses cliqueMS to generate components. This function is called when calling `generateComponents` with `algorithm="cliquems"`.

The grouping of features in each component ('clique') is based on high similarity of chromatographic elution profiles. All features in each component are then annotated with the `cliqueMS::getIsotopes` and `cliqueMS::getAnnotation` functions.

Value

A `componentsFeatures` derived object.

Feature components

The returned components are based on so called *feature components*. Unlike other algorithms, components are first made on a feature level (per analysis), instead of for complete feature groups. In the final step the feature components are converted to 'regular' components by employing a consensus approach with the following steps:

1. If an adduct assigned to a feature only occurs as a minority compared to other adduct assignments within the same feature group, it is considered as an outlier and removed accordingly (controlled by the `relMinAdductAbundance` argument).
2. For features within a feature group, only keep their adduct assignment if it occurs as the most frequent or is preferential (controlled by `adductConflictsUsePref` and `prefAdducts` arguments).
3. Components are made by combining the feature groups for which at least one of their features are jointly present in the same feature component.
4. Conflicts of neutral mass assignments within a component (*i.e.* not all are the same) are dealt with. Firstly, all feature groups with an unknown neutral mass are split in another component. Then, if conflicts still occur, the feature groups with similar neutral mass (determined by `absMzDev` argument) are grouped. Depending on the `NMConflicts` argument, the group with one or more preferential adduct(s) or that is the largest or most intense is selected, whereas others are removed from the component. In case multiple groups contain preferential adducts, and '>1' preferential adducts are available, the group with the adduct that matches first in `prefAdducts` 'wins'. In case of ties, one of the next strategies in `NMConflicts` is tried.
5. If a feature group occurs in multiple components it will be removed completely.
6. the `minSize` filter is applied.

Sets workflows

In a sets workflow the componentization is first performed for each set independently. The resulting components are then all combined in a `componentsSet` object. Note that the components themselves are never merged. The components are renamed to include the set name from which they were generated (*e.g.* `"CMP1"` becomes `"CMP1-positive"`).

References

Senan O, Aguilar-Mogas A, Navarro M, Capellades J, Noon L, Burks D, Yanes O, Guimera R, Sales-Pardo M (2019). "CliqueMS: a computational tool for annotating in-source metabolite ions from LC-MS untargeted metabolomics data based on a coelution similarity network." *Bioinformatics*, **35**(20), 4089–4097. doi:10.1093/bioinformatics/btz207.

See Also

`generateComponents` for more details and other algorithms.

`generateComponentsIntClust` *Generate components based on intensity profiles*

Description

Generates components based on intensity profiles of feature groups.

Usage

```
generateComponentsIntClust(fGroups, ...)

## S4 method for signature 'featureGroups'
generateComponentsIntClust(
  fGroups,
  method = "complete",
  metric = "euclidean",
  normalized = TRUE,
  average = TRUE,
  maxTreeHeight = 1,
  deepSplit = TRUE,
  minModuleSize = 1
)
```

Arguments

fGroups	featureGroups object for which components should be generated.
...	Any parameters to be passed to the selected component generation algorithm.
method	Clustering method that should be applied (passed to fastcluster::hclust).
metric	Distance metric used to calculate the distance matrix (passed to daisy).
normalized, average	Passed to as.data.table to perform normalization and averaging of data.
maxTreeHeight, deepSplit, minModuleSize	Arguments used by cutreeDynamicTree.

Details

This function uses hierarchical clustering of intensity profiles to generate components. This function is called when calling generateComponents with algorithm="intclust".

Hierarchical clustering is performed on normalized (and optionally replicate averaged) intensity data and the resulting dendrogram is automatically cut with cutreeDynamicTree. The distance matrix is calculated with daisy and clustering is performed with fastcluster::hclust. The clustering of the resulting components can be further visualized and modified using the methods defined for componentsIntClust.

Value

The components are stored in objects derived from componentsIntClust.

Sets workflows

In a sets workflow normalization of feature intensities occur per set.

References

Müllner D (2013). "fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python." *Journal of Statistical Software*, **53**(9), 1–18. doi:10.18637/jss.v053.i09.

Schollee JE, Bourgin M, von Gunten U, McArdell CS, Hollender J (2018). "Non-target screening to trace ozonation transformation products in a wastewater treatment train including different post-treatments." *Water Research*, **142**, 267–278. doi:10.1016/j.watres.2018.05.045.

See Also

`generateComponents` for more details and other algorithms.

`generateComponentsNontarget` *Componentization of homologous series with nontarget*

Description

Uses the nontarget R package to generate components by unsupervised detection of homologous series.

Usage

```
generateComponentsNontarget(fGroups, ...)

## S4 method for signature 'featureGroups'
generateComponentsNontarget(
  fGroups,
  ionization = NULL,
  rtRange = c(-120, 120),
  mzRange = c(5, 120),
  elements = c("C", "H", "O"),
  rtDev = 30,
  absMzDev = 0.002,
  absMzDevLink = absMzDev * 2,
  traceHack = all(R.Version()[c("major", "minor")] >= c(3, 4)),
  ...
)

## S4 method for signature 'featureGroupsSet'
generateComponentsNontarget(fGroups, ionization = NULL, ...)
```

Arguments

`fGroups`	`featureGroups` object for which components should be generated.
`...`	Any further arguments passed to `homol.search`.
	(**sets workflow**) Further arguments passed to the non-sets workflow method.

ionization	Which ionization polarity was used to generate the data: should be "positive" or "negative". If the featureGroups object has adduct annotations, and ionization=NULL, the ionization will be detected automatically. (**sets workflow**) This parameter is not supported for sets workflows, as the ionization will always be detected automatically.
rtRange	A numeric vector containing the minimum and maximum retention time (in seconds) between homologues. Series are always considered from low to high *m/z*, thus, a negative minimum retention time allows detection of homologous series with increasing *m/z* and decreasing retention times. These values set the minrt and maxrt arguments of homol.search.
mzRange	A numeric vector specifying the minimum and maximum *m/z* increment of a homologous series. Sets the minmz and maxmz arguments of homol.search.
elements	A character vector with elements to be considered for detection of repeating units. Sets the elements argument of homol.search function.
rtDev	Maximum retention time deviation. Sets the rttol to homol.search.
absMzDev	Maximum absolute *m/z* deviation. Sets the mztol argument to homol.search
absMzDevLink	Maximum absolute *m/z* deviation when linking series. This should usually be a bit higher than absMzDev to ensure proper linkage.
traceHack	Currently homol.search does not work with R '>3.3.3'. This flag, which is enabled by default on these R versions, implements a (messy) workaround (more details here).

Details

This function uses nontarget to generate components. This function is called when calling generateComponents with algorithm="nontarget".

In the first step the homol.search function is used to detect all homologous series within each replicate group (analyses within each replicate group are averaged prior to detection). Then, homologous series across replicate groups are merged in case of full overlap or when merging of partial overlapping series causes no conflicts.

Value

The generated comnponents are returned as an object from the componentsNT class.

Sets workflows

In a sets workflow the componentization is first performed for each set independently. The resulting components are then all combined in a componentsNTSet object. Note that the components themselves are never merged. The components are renamed to include the set name from which they were generated (*e.g.* "CMP1" becomes "CMP1-positive").

The output class supports additional methods such as plotGraph.

References

Loos M, Singer H (2017). "Nontargeted homologue series extraction from hyphenated high resolution mass spectrometry data." *Journal of Cheminformatics*, **9**(1). doi:10.1186/s133210170197z.

Loos M, Gerber C, Corona F, Hollender J, Singer H (2015). "Accelerated Isotope Fine Structure Calculation Using Pruned Transition Trees." *Analytical Chemistry*, **87**(11), 5738-5744. https://pubs.acs.org/doi/abs/10.1021/acs.analchem.5b00941.

See Also

generateComponents for more details and other algorithms.

`generateComponentsOpenMS` *Componentization of adducts, isotopes etc. with OpenMS*

Description

Uses the MetaboliteAdductDecharger utility (see http://www.openms.de) to generate components.

Usage

```
generateComponentsOpenMS(fGroups, ...)

## S4 method for signature 'featureGroups'
generateComponentsOpenMS(
  fGroups,
  ionization = NULL,
  chargeMin = 1,
  chargeMax = 1,
  chargeSpan = 3,
  qTry = "heuristic",
  potentialAdducts = NULL,
  minRTOverlap = 0.66,
  retWindow = 1,
  absMzDev = 0.005,
  minSize = 2,
  relMinAdductAbundance = 0.75,
  adductConflictsUsePref = TRUE,
  NMConflicts = c("preferential", "mostAbundant", "mostIntense"),
  prefAdducts = c("[M+H]+", "[M-H]-"),
  extraOpts = NULL
)

## S4 method for signature 'featureGroupsSet'
generateComponentsOpenMS(
```

```
  fGroups,
  ionization = NULL,
  chargeMin = 1,
  chargeMax = 1,
  chargeSpan = 3,
  qTry = "heuristic",
  potentialAdducts = NULL,
  ...
)
```

Arguments

fGroups	featureGroups object for which components should be generated.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
ionization	Which ionization polarity was used to generate the data: should be "positive" or "negative". If the featureGroups object has adduct annotations, and ionization=NULL, the ionization will be detected automatically. (**sets workflow**) This parameter is not supported for sets workflows, as the ionization will always be detected automatically.
chargeMin, chargeMax	The minimum/maximum charge to consider. Corresponds to the algorithm:MetaboliteFeatureDeconv options.
chargeSpan	The maximum charge span for a single analyte. Corresponds to algorithm:MetaboliteFeatureDeconvo
qTry	Sets how charges are determined. Corresponds to algorithm:MetaboliteFeatureDeconvolution:q_tr Valid options are "heuristic" and "all" (the "feature" option from OpenMS is currently not supported).
potentialAdducts	The adducts to consider. Should be a numeric vector with probabilities for each adduct, *e.g.* potentialAdducts=c("[M+H]+" = 0.8, "[M+Na]+" = 0.2). Note that the sum of probabilities should always be '1'. Furthermore, note that additions of multiple adducts should be controlled by the chargeMin/chargeMax arguments (and *not* with potentialAdducts), *e.g.* if chargeMax=2 then both [M+H]+ and [2M+H]2+ may be considered. Please see the algorithm:MetaboliteFeatureDeconvolutio option of MetaboliteAdductDecharger for more details. If NULL then the a default is chosen with defaultOpenMSAdducts (which is *not* the same as OpenMS). (**sets workflow**) Should be a list where each entry specifies the potential adducts for a set. Should either be named with the sets names or follow the same order as sets(fGroups). Example: potentialAdducts=list(positive=c("[M+H]+" = 0.8, "[M+Na]+" = 0.2), negative=c("[M-H]-" = 0.8, "[M-H2O-H]-" = 0.2))
minRTOverlap, retWindow	Sets feature retention tolerances when grouping features. Sets the "algorithm:MetaboliteFeatureDeco and algorithm:MetaboliteFeatureDeconvolution:min_rt_overlap options.
absMzDev	Maximum absolute *m/z* deviation. Sets the algorithm:MetaboliteFeatureDeconvolution:mass_max_ option
minSize	The minimum size of a component. Smaller components than this size will be removed. See note below.

`relMinAdductAbundance`
: The minimum relative abundance (‘`0-1`’) that an adduct should be assigned to features within the same feature group. See the `Feature components` section for more details.

`adductConflictsUsePref`
: If set to `TRUE`, and not all adduct assigments to the features within a feature group are equal and at least one of those adducts is a preferential adduct (`prefAdducts` argument), then only the features with (the lowest ranked) preferential adduct are considered. In all other cases or when `adductConflictsUsePref=FALSE` only features with the most frequently assigned adduct is considered. See the `Feature components` section for more details.

`NMConflicts`
: The strategies to employ when not all neutral masses within a component are equal. Valid options are: `"preferential"`, `"mostAbundant"` and `"mostIntense"`. Multiple strategies are possible, and will be executed in the given order until one succeeds. See the `Feature components` section for more details.

`prefAdducts`
: A `character` vector with one or more *preferential adducts*. See the `Feature components` section for more details.

`extraOpts`
: Named character vector with extra command line parameters directly passed to `MetaboliteAdductDecharger`. Set to `NULL` to ignore.

Details

This function uses OpenMS to generate components. This function is called when calling `generateComponents` with `algorithm="openms"`.

Features that show highly similar chromatographic elution profiles are grouped, and subsequently annotated with their adducts.

Value

A `componentsFeatures` derived object.

Feature components

The returned components are based on so called *feature components*. Unlike other algorithms, components are first made on a feature level (per analysis), instead of for complete feature groups. In the final step the feature components are converted to ’regular’ components by employing a consensus approach with the following steps:

1. If an adduct assigned to a feature only occurs as a minority compared to other adduct assigments within the same feature group, it is considered as an outlier and removed accordingly (controlled by the `relMinAdductAbundance` argument).
2. For features within a feature group, only keep their adduct assignment if it occurs as the most frequent or is preferential (controlled by `adductConflictsUsePref` and `prefAdducts` arguments).
3. Components are made by combining the feature groups for which at least one of their features are jointly present in the same feature component.

4. Conflicts of neutral mass assignments within a component (*i.e.* not all are the same) are dealt with. Firstly, all feature groups with an unknown neutral mass are split in another component. Then, if conflicts still occur, the feature groups with similar neutral mass (determined by `absMzDev` argument) are grouped. Depending on the `NMConflicts` argument, the group with one or more preferential adduct(s) or that is the largest or most intense is selected, whereas others are removed from the component. In case multiple groups contain preferential adducts, and '>1' preferential adducts are available, the group with the adduct that matches first in `prefAdducts` 'wins'. In case of ties, one of the next strategies in `NMConflicts` is tried.
5. If a feature group occurs in multiple components it will be removed completely.
6. the `minSize` filter is applied.

Sets workflows

In a sets workflow the componentization is first performed for each set independently. The resulting components are then all combined in a `componentsSet` object. Note that the components themselves are never merged. The components are renamed to include the set name from which they were generated (*e.g.* `"CMP1"` becomes `"CMP1-positive"`).

Parallelization

generateComponentsOpenMS uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

References

Bielow C, Ruzek S, Huber CG, Reinert K (2010). "Optimal Decharging and Clustering of Charge Ladders Generated in ESI-MS." *Journal of Proteome Research*, **9**(5), 2688–2695. doi:10.1021/pr100177k.

See Also

`generateComponents` for more details and other algorithms.

`generateComponentsRAMClustR` *Componentization of adducts, isotopes etc. with RAMClustR*

Description

Uses RAMClustR to generate components from feature groups which follow similar chromatographic retention profiles and annotate their relationships (*e.g.* adducts and isotopes).

Usage

```
generateComponentsRAMClustR(fGroups, ...)

## S4 method for signature 'featureGroups'
generateComponentsRAMClustR(
  fGroups,
  ionization = NULL,
  st = NULL,
  sr = NULL,
  maxt = 12,
  hmax = 0.3,
  normalize = "TIC",
  absMzDev = 0.002,
  relMzDev = 5,
  minSize = 2,
  relMinReplicates = 0.5,
 RCExperimentVals = list(design = list(platform = "LC-MS"), instrument = list(ionization
    = ionization, MSlevs = 1)),
  extraOptsRC = NULL,
  extraOptsFM = NULL
)

## S4 method for signature 'featureGroupsSet'
generateComponentsRAMClustR(fGroups, ionization = NULL, ...)
```

Arguments

fGroups	featureGroups object for which components should be generated.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
ionization	Which ionization polarity was used to generate the data: should be "positive" or "negative". If the featureGroups object has adduct annotations, and ionization=NULL, the ionization will be detected automatically. (**sets workflow**) This parameter is not supported for sets workflows, as the ionization will always be detected automatically.
st, sr, maxt, hmax, normalize	Arguments to tune the behaviour of feature group clustering. See their documentation from ramclustR. When st is NULL it will be automatically calculated as the half of the median for all chromatographic peak widths.
absMzDev	Maximum absolute *m/z* deviation. Sets the mzabs.error argument to do.findmain
relMzDev	Maximum relative mass deviation (PPM). Sets the ppm.error argument to do.findmain.
minSize	The minimum size of a component. Smaller components than this size will be removed. See note below. Sets the minModuleSize argument to ramclustR.
relMinReplicates	Feature groups within a component are only kept when they contain data for at least this (relative) amount of replicate analyses. For instance, '0.5' means

that at least half of the replicates should contain data for a particular feature group in a component. In this calculation replicates that are fully absent within a component are not taken in to account. See note below.

`RCExperimentVals`
: A named `list` containing two more `lists`: `design` and `instrument`. These are used to construct the `ExpDes` argument passed to `ramclustR`.

`extraOptsRC`, `extraOptsFM`
: Named `list` with further arguments to be passed to `ramclustR` and `do.findmain`. Set to `NULL` to ignore.

Details

This function uses RAMClustR to generate components. This function is called when calling `generateComponents` with `algorithm="ramclustr"`.

This method uses the `ramclustR` functions for generating the components, whereas `do.findmain` is used for annotation.

Value

A `components` (derived) object containing all generated components.

Sets workflows

In a sets workflow the componentization is first performed for each set independently. The resulting components are then all combined in a `componentsSet` object. Note that the components themselves are never merged. The components are renamed to include the set name from which they were generated (*e.g.* `"CMP1"` becomes `"CMP1-positive"`).

Note

The default value for `relMinReplicates` results in extra filtering, hence, the final results may be different than what the algorithm normally would return.

References

Broeckling, Heuberger CD;, Prince AL;, Ingelsson JA;, Prenni E;, E. J (2013). "Assigning precursor-product ion relationships in indiscriminant MS/MS data from non-targeted metabolite profiling studies." *Analytical Chemistry*, **9**, 33-43.

Broeckling CD, Afsar FA, Neumann S, Ben-Hur A, Prenni JE (2014). "RAMClust: A Novel Feature Clustering Method Enables Spectral-Matching-Based Annotation for Metabolomics Data." *Analytical Chemistry*, **86 (14)**, 6812–6817.

See Also

`generateComponents` for more details and other algorithms.

`generateComponentsSpecClust` *Generate components based on MS/MS similarity*

Description

Generates components based on MS/MS similarity between feature groups.

Usage

```
generateComponentsSpecClust(fGroups, ...)

## S4 method for signature 'featureGroups'
generateComponentsSpecClust(
  fGroups,
  MSPeakLists,
  method = "complete",
  specSimParams = getDefSpecSimParams(),
  maxTreeHeight = 1,
  deepSplit = TRUE,
  minModuleSize = 1
)
```

Arguments

`fGroups`	`featureGroups` object for which components should be generated.
`...`	Any parameters to be passed to the selected component generation algorithm.
`MSPeakLists`	The `MSPeakLists` object for the given feature groups that should be used for MS spectral similarity calculations.
`method`	Clustering method that should be applied (passed to `fastcluster::hclust`).
`specSimParams`	A named `list` with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.
`maxTreeHeight, deepSplit, minModuleSize`	Arguments used by `cutreeDynamicTree`.

Details

This function uses hierarchical clustering of MS/MS spectra to generate components. This function is called when calling `generateComponents` with `algorithm="specclust"`.

The similarities are converted to a distance matrix and used as input for hierarchical clustering, and the resulting dendrogram is automatically cut with `cutreeDynamicTree`. The clustering is performed with `fastcluster::hclust`.

Value

The components are stored in objects derived from `componentsSpecClust`.

Sets workflows

In a sets workflow the spectral similarities for each set are combined as is described for the `spectrumSimilarity` method for sets workflows.

Author(s)

Rick Helmus <<r.helmus@uva.nl>> and Bas van de Velde (major contributions to spectral binning and similarity calculation).

References

Müllner D (2013). "fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python." *Journal of Statistical Software*, **53**(9), 1–18. doi:10.18637/jss.v053.i09.

See Also

`generateComponents` for more details and other algorithms.

`generateComponentsTPs` *Generate components of transformation products*

Description

Generates components by linking feature groups of transformation products and their parents.

Usage

```
generateComponentsTPs(fGroups, ...)

## S4 method for signature 'featureGroups'
generateComponentsTPs(
  fGroups,
  fGroupsTPs = fGroups,
  ignoreParents = FALSE,
  TPs = NULL,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  minRTDiff = 20,
  specSimParams = getDefSpecSimParams()
)

## S4 method for signature 'featureGroupsSet'
generateComponentsTPs(
  fGroups,
  fGroupsTPs = fGroups,
  ignoreParents = FALSE,
```

```
  TPs = NULL,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  minRTDiff = 20,
  specSimParams = getDefSpecSimParams()
)
```

Arguments

fGroups	The input featureGroups for componentization. See fGroupsTPs.
...	Further arguments specified to the methods.
fGroupsTPs	A featureGroups object containing the feature groups that are expected to be transformation products. If a distinction between parents and TPs is not yet known, fGroupsTPs should equal the fGroups argument. Otherwise, fGroups should only contain the parent feature groups, and both fGroups and fGroupsTPs *must* be a subset of the same featureGroups object.
ignoreParents	If TRUE then feature groups present in both fGroups and fGroupsTPs are not considered as TPs.
TPs	A transformationProducts object. Set to NULL to perform linking without this data.
MSPeakLists, formulas, compounds	A MSPeakLists/formulas/compounds object to calculate MS/MS or annotation similarities between parents and TPs. If NULL then this data is not calculated. For more details see the Linking parents and transformation products section below.
minRTDiff	Minimum retention time (in seconds) difference between the parent and a TP to determine whether a TP elutes prior/after the parent (to calculate retDir values, see Details in componentsTPs))
specSimParams	A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.

Details

This function uses transformation product screening to generate components. This function is called when calling generateComponents with algorithm="tp".

This method typically employs data from generated transformation products to find parents and their TPs. However, this data is not necessary, and components can also be made based on MS/MS similarity and/or other annotation similarities between the parent and its TPs. For more details see the Linking parents and transformation products section below.

Value

The components are stored in objects derived from componentsTPs.

Linking parents and transformation products

Each component consists of feature groups that are considered to be transformation products for one parent (the parent that 'belongs' to the component can be retrieved with the componentInfo method). The parent feature groups are taken from the fGroups parameter, while the feature groups for TPs are taken from fGroupsTPs. If a feature group occurs in both variables, it may therefore be considered as both a parent or TP.

If transformation product data is given, *i.e.* the TPs argument is set, then a suspect screening of the TPs must be performed in advance (see screenSuspects and convertToSuspects to create the suspect list). Furthermore, if TPs were generated with generateTPsBioTransformer or generateTPsLibrary then the suspect screening must also include the parents (*e.g.* by setting includeParents=TRUE when calling convertToSuspects or by amending results by setting amend=TRUE to screenSuspects). The suspect screening is necessary for the componentization algorithm to map the feature groups of the parent or TP. If the the suspect screening yields multiple TP hits, all will be reported. Similarly, if the suspect screening contains multiple hits for a parent, a component is made for each of the parent hits.

In case no transformation product data is provided (TPs=NULL), the componentization algorithm simply assumes that each feature group from fGroupsTPs is a potential TP for every parent feature group in fGroups. For this reason, it is highly recommended to specify which feature groups are parents/TPs (see the fGroupsTPs argument description above) and *crucial* that the data is post-processed, for instance by only retaining TPs that have high annotation similarity with their parents (see the filter method for componentsTPs).

A typical way to distinguish which feature groups are parents or TPs from two different (groups of) samples is by calculating Fold Changes (see the as.data.table method for feature groups and plotVolcano). Of course, other statistical techniques from R are also suitable.

During componentization, several characteristics are calculated which may be useful for post-processing:

- specSimilarity: the MS/MS spectral similarity between the feature groups of the TP and its parent ('0-1').
- specSimilarityPrec,specSimilarityBoth: as specSimilarity, but calculated with binned data using the "precursor" and "both" method, respectively (see MS spectral similarity parameters for more details).
- fragmentMatches The number of MS/MS fragment formula annotations that overlap between the TP and parent. If both the formulas and compounds arguments are specified then the annotation data is pooled prior to calculation. Note that only unique matches are counted. Furthermore, note that annotations from *all* candidates are considered, even if the formula/structure of the parent/TP is known. Hence, fragmentMatches is mainly useful when little or no chemical information is known on the parents/TPs, *i.e.*, when TPs=NULL or originates from generateTPsLogic. Since annotations for all candidates are used, it is highly recommended that the annotation objects are first processed with the filter method, for instance, to select only the top ranked candidates.
- neutralLossMatches As fragmentMatches, but counting overlapping neutral loss formulae.
- retDir The retention time direction of the TP relative to its parent. See Details in componentsTPs. If TP data was specified, the expected direction is stored in TP_retDir.
- retDiff,mzDiff,formulaDiff The retention time, *m/z* and formula difference between the parent and TP (latter only available if data TP formula is available).

Sets workflows

In a sets workflow the component tables are amended with extra information such as overall/specific set spectrum similarities. As sets data is mixed, transformation products are able to be linked with a parent, even if they were not measured in the same set.

Note

The `shift` parameter of `specSimParams` is ignored by `generateComponentsTPs`, since it always calculates similarities with all supported options.

See Also

`generateComponents` for more details and other algorithms.

`generateCompounds` *Automatic compound annotation*

Description

Automatically perform chemical compound annotation for feature groups.

Usage

```
generateCompounds(fGroups, MSPeakLists, algorithm, ...)

## S4 method for signature 'featureGroups'
generateCompounds(fGroups, MSPeakLists, algorithm, ...)
```

Arguments

`fGroups`	`featureGroups` object which should be annotated. This should be the same or a subset of the object that was used to create the specified `MSPeakLists`. In the case of a subset only the remaining feature groups in the subset are considered.
`MSPeakLists`	A `MSPeakLists` object that was generated for the supplied `fGroups`.
`algorithm`	A character string describing the algorithm that should be used: `"metfrag"`, `"sirius"`, `"library"`
`...`	Any parameters to be passed to the selected compound generation algorithm.

Details

Several algorithms are provided to automatically perform compound annotation for feature groups. To this end, measured masses for all feature groups are searched within online database(s) (*e.g.* PubChem) to retrieve a list of potential candidate chemical compounds. Depending on the algorithm and its parameters, further scoring of candidates is then performed using, for instance, matching of measured and theoretical isotopic patterns, presence within other data sources such as patent databases and similarity of measured and in-silico predicted MS/MS fragments. Note that this

process is often quite time consuming, especially for large feature group sets. Therefore, this is often one of the last steps within the workflow and not performed before feature groups have been prioritized.

`generateCompounds` is a generic function that will generateCompounds by one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as `generateCompoundsMetFrag` and `generateCompoundsSIRIUS`. While these functions may be called directly, `generateCompounds` provides a generic interface and is therefore usually preferred.

Value

A `compounds` derived object containing all compound annotations.

Scorings

Each algorithm implements their own scoring system. Their names have been simplified and harmonized where possible. The `compoundScorings` function can be used to get an overview of both the algorithm specific and generic scoring names.

Sets workflows

With a sets workflow, annotation is first performed for each set. This is important, since the annotation algorithms typically cannot work with data from mixed ionization modes. The annotation results are then combined to generate a *sets consensus*:

- The annotation tables for each feature group from the set specific data are combined. Rows with overlapping candidates (determined by the first-block INCHIKEY) are merged.
- Set specific data (*e.g.* the ionic formula) is retained by renaming their columns with set specific names.
- The MS/MS fragment annotations (`fragInfo` column) from each set are combined.
- The scorings for each set are averaged to calculate overall scores. if `setAvgSpecificScores=FALSE` then scorings that are considered set specific (*e.g.* MS/MS and isotopic pattern match) are *not* averaged.
- The candidates are re-ranked based on their average ranking among the set data (if a candidate is absent in a set it is assigned the poorest rank in that set).
- The coverage of each candidate among sets is calculated. Depending on the `setThreshold` and `setThresholdAnn` arguments, candidates with low abundance are removed.

See Also

The `compounds` output class and its methods and the algorithm specific functions: `generateCompoundsMetFrag`, `generateCompoundsSIRIUS`, `generateCompoundsLibrary`

generateCompoundsLibrary *Compound annotation with an MS library*

Description

Uses a MS library loaded by loadMSLibrary for compound annotation.

Usage

```
generateCompoundsLibrary(fGroups, ...)

## S4 method for signature 'featureGroups'
generateCompoundsLibrary(
  fGroups,
  MSPeakLists,
  MSLibrary,
  minSim = 0.75,
  minAnnSim = minSim,
  absMzDev = 0.002,
  adduct = NULL,
  checkIons = "adduct",
  spectrumType = "MS2",
  specSimParams = getDefSpecSimParams(),
  specSimParamsLib = getDefSpecSimParams()
)

## S4 method for signature 'featureGroupsSet'
generateCompoundsLibrary(
  fGroups,
  MSPeakLists,
  MSLibrary,
  minSim = 0.75,
  minAnnSim = minSim,
  absMzDev = 0.002,
  adduct = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0,
  setAvgSpecificScores = FALSE
)
```

Arguments

fGroups	featureGroups object which should be annotated. This should be the same or a subset of the object that was used to create the specified MSPeakLists. In the case of a subset only the remaining feature groups in the subset are considered.

`...` (**sets workflow**) Further arguments passed to the non-sets workflow method.

`MSPeakLists` A `MSPeakLists` object that was generated for the supplied `fGroups`.

`MSLibrary` The `MSLibrary` object that should be used to find candidates.

`minSim` The minimum spectral similarity for candidate records.

`minAnnSim` The minimum spectral similarity of a record for it to be used to find annotations (see the `Details` section).

`absMzDev` The maximum absolute *m/z* deviation between the feature group and library record *m/z* values for candidate selection.

`adduct` An `adduct` object (or something that can be converted to it with `as.adduct`). Examples: `"[M-H]-"`, `"[M+Na]+"`. If the `featureGroups` object has adduct annotations then these are used if `adducts=NULL`.

(**sets workflow**) The `adduct` argument is not supported for sets workflows, since the adduct annotations will then always be used.

`checkIons` A `character` that excludes library records with different adduct (`checkIons="adduct"`) or MS ionization polarity (`checkIons="polarity"`). If `checkIons="none"` then these filters are not applied.

`spectrumType` A `character` vector which limits library records to the given spectrum types (`Spectrum_type` field, *e.g.* `"MS2"`). Set to `NULL` to allow all spectrum types.

`specSimParams` A named `list` with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.

`specSimParamsLib`

Like `specSimParams`, but these parameters *only* influence pre-treatment of library spectra (only the `removePrecursor`, `relMinIntensity` and `minPeaks` parameters are used).

`setThreshold` (**sets workflow**) Minimum abundance for a candidate among all sets (‘`0-1`’). For instance, a value of ‘`1`’ means that the candidate needs to be present in all the set data.

`setThresholdAnn`

(**sets workflow**) As `setThreshold`, but only taking into account the set data that contain annotations for the feature group of the candidate.

`setAvgSpecificScores`

(**sets workflow**) If `TRUE` then set specific scorings (*e.g.* MS/MS match) are also averaged.

Details

This function uses MS library spectra to generate compound candidates. This function is called when calling `generateCompounds` with `algorithm="library"`.

This method matches measured MS/MS data (peak lists) with those from an MS library to find candidate structures. Hence, only feature groups with MS/MS peak list data are annotated.

The library is searched for candidates with the following criteria:

1. Only records with ion *m/z* (`PrecursorMZ`), SMILES, INCHI, INCHIKEY and `formula` data are considered.

2. Depending on the value of the `checkIons` argument, records with different adduct (`Precursor_type`) or polarity (`Ion_mode`) may be ignored.
3. The *m/z* values of the candidate and feature group should match (tolerance set by `absMzDev` argument).
4. The spectral similarity should not be lower than the value defined for the `minSim` argument.
5. If multiple candidates with the same first-block INCHIKEY are found then only the candidate with the best spectral match is kept.

If the library contains annotations these will be added to the matched MS/MS peaks. However, since the candidate selected from criterion #5 above may not contain all the annotation data available from the MS library, annotations from other records are also considered (controlled by the `minAnnSim` argument). If this leads to different annotations for the same mass peak then only the most abundant annotation is kept.

See Also

`generateCompounds` for more details and other algorithms.

`loadMSLibrary` to obtain MS library data and the methods for `MSLibrary` to treat the data before using it for annotation.

`generateCompoundsMetFrag` *Compound annotation with MetFrag*

Description

Uses the **metfRag** package or `MetFrag CL` for compound identification (see `http://ipb-halle.github.io/MetFrag/`).

Usage

```
generateCompoundsMetFrag(fGroups, ...)

## S4 method for signature 'featureGroups'
generateCompoundsMetFrag(
  fGroups,
  MSPeakLists,
  method = "CL",
  timeout = 300,
  timeoutRetries = 2,
  errorRetries = 2,
  topMost = 100,
  dbRelMzDev = 5,
  fragRelMzDev = 5,
  fragAbsMzDev = 0.002,
  adduct = NULL,
```

```
  database = "pubchem",
  extendedPubChem = "auto",
  chemSpiderToken = "",
  scoreTypes = compoundScorings("metfrag", database, onlyDefault = TRUE)$name,
  scoreWeights = 1,
  preProcessingFilters = c("UnconnectedCompoundFilter", "IsotopeFilter"),
  postProcessingFilters = c("InChIKeyFilter"),
  maxCandidatesToStop = 2500,
  identifiers = NULL,
  extraOpts = NULL
)

## S4 method for signature 'featureGroupsSet'
generateCompoundsMetFrag(
  fGroups,
  MSPeakLists,
  method = "CL",
  timeout = 300,
  timeoutRetries = 2,
  errorRetries = 2,
  topMost = 100,
  dbRelMzDev = 5,
  fragRelMzDev = 5,
  fragAbsMzDev = 0.002,
  adduct = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0,
  setAvgSpecificScores = FALSE
)
```

Arguments

fGroups	featureGroups object which should be annotated. This should be the same or a subset of the object that was used to create the specified MSPeakLists. In the case of a subset only the remaining feature groups in the subset are considered.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
MSPeakLists	A MSPeakLists object that was generated for the supplied fGroups.
method	Which method should be used for MetFrag execution: "CL" for MetFragCL and "R" for MetFragR. The former is usually much faster and recommended.
timeout	Maximum time (in seconds) before a metFrag query for a feature group is stopped. Also see timeoutRetries argument.
timeoutRetries	Maximum number of retries after reaching a timeout before completely skipping the metFrag query for a feature group. Also see timeout argument.
errorRetries	Maximum number of retries after an error occurred. This may be useful to handle e.g. connection errors.

topMost — Only keep this number of candidates (per feature group) with highest score. Set to NULL to always keep all candidates, however, please note that this may result in significant usage of CPU/RAM resources for large numbers of candidates.

dbRelMzDev — Relative mass deviation (in ppm) for database search. Sets the 'DatabaseSearchRelativeMassDeviatio option.

fragRelMzDev — Relative mass deviation (in ppm) for fragment matching. Sets the 'FragmentPeakMatchRelativeMassDev option.

fragAbsMzDev — Absolute mass deviation (in Da) for fragment matching. Sets the 'FragmentPeakMatchAbsoluteMassDev option.

adduct — An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+". If the featureGroups object has adduct annotations then these are used if adducts=NULL.

(**sets workflow**) The adduct argument is not supported for sets workflows, since the adduct annotations will then always be used.

database — Compound database to use. Valid values are: "pubchem", "chemspider", "for-ident", "comptox", "pubchemlite", "kegg", "sdf", "psv" and "csv". See section below for more information. Sets the MetFragDatabaseType option.

extendedPubChem — If database="pubchem": whether to use the *extended* database that includes information for compound scoring (*i.e.* number of patents/PubMed references). Note that downloading candidates from this database might take extra time. Valid values are: FALSE (never use it), TRUE (always use it) or "auto" (default, use if specified scorings demand it).

chemSpiderToken — A character string with the ChemSpider security token that should be set when the ChemSpider database is used. Sets the 'ChemSpiderToken' option.

scoreTypes — A character vector defining the scoring types. See the Scorings section below for more information. Note that both generic and MetFrag specific names are accepted (*i.e.* name and metfrag columns returned by compoundScorings). When a local database is used, the name should match what is given there (e.g column names when database=csv). Note that MetFrag may still report other scoring data, however, these are not used for ranking. Sets the 'MetFragScoreTypes' option.

scoreWeights — Numeric vector containing weights of the used scoring types. Order is the same as set in scoreTypes. Values are recycled if necessary. Sets the 'MetFragScoreWeights' option.

preProcessingFilters, postProcessingFilters — A character vector defining pre/post filters applied before/after fragmentation and scoring (*e.g.* "UnconnectedCompoundFilter", "IsotopeFilter", "ElementExclusionFilter"). Some methods require further options to be set. For all filters and more information refer to the Candidate Filters section on the MetFragR homepage. Sets the 'MetFragPreProcessingCandidateFilter' and MetFragPostProcessingCandidateFilter options.

maxCandidatesToStop — If more than this number of candidate structures are found then processing will be aborted and no results this feature group will be reported. Low values increase the chance of missing data, whereas too high values will use too much

computer resources and signficantly slowdown the process. Sets the 'MaxCandidateLimitToStop' option.

identifiers A list containing for each feature group a character vector with database identifiers that should be used to find candidates for a feature group (the list should be named by feature group names). If NULL all relevant candidates will be retrieved from the specified database. An example usage scenario is to obtain the list of candidate identifiers from a compounds object obtained with generateCompoundsSIRIUS using the identifiers method. This way, only those candidates will be searched by MetFrag that were generated by SIRIUS+CSI:FingerID. Sets the 'PrecursorCompoundIDs' option.

extraOpts A named list containing further settings MetFrag. See the MetFragR and MetFrag CL homepages for all available options. Set to NULL to ignore.

setThreshold (**sets workflow**) Minimum abundance for a candidate among all sets ('0-1'). For instance, a value of '1' means that the candidate needs to be present in all the set data.

setThresholdAnn (**sets workflow**) As setThreshold, but only taking into account the set data that contain annotations for the feature group of the candidate.

setAvgSpecificScores (**sets workflow**) If TRUE then set specific scorings (*e.g.* MS/MS match) are also averaged.

Details

This function uses MetFrag to generate compound candidates. This function is called when calling generateCompounds with algorithm="metfrag".

Several online compound databases such as PubChem and ChemSpider may be chosen for retrieval of candidate structures. This method requires the availability of MS/MS data, and feature groups without it will be ignored. Many options exist to score and filter resulting data, and it is highly suggested to optimize these to improve results. The MetFrag options PeakList, IonizedPrecursorMass and ExperimentalRetentionTimeValue (in minutes) fields are automatically set from feature data.

Value

generateCompoundsMetFrag returns a compoundsMF object.

Scorings

MetFrag supports *many* different scorings to rank candidates. The compoundScorings function can be used to get an overview: (some columns are omitted)

name	metfrag	database
score	Score	
fragScore	FragmenterScore	
metFusionScore	OfflineMetFusionScore	
individualMoNAScore	OfflineIndividualMoNAScore	

numberPatents	PubChemNumberPatents	pubchem
numberPatents	Patent_Count	pubchemlite
pubMedReferences	PubChemNumberPubMedReferences	pubchem
pubMedReferences	ChemSpiderNumberPubMedReferences	chemspider
pubMedReferences	NUMBER_OF_PUBMED_ARTICLES	comptox
pubMedReferences	PubMed_Count	pubchemlite
extReferenceCount	ChemSpiderNumberExternalReferences	chemspider
dataSourceCount	ChemSpiderDataSourceCount	chemspider
referenceCount	ChemSpiderReferenceCount	chemspider
RSCCount	ChemSpiderRSCCount	chemspider
smartsInclusionScore	SmartsSubstructureInclusionScore	
smartsExclusionScore	SmartsSubstructureExclusionScore	
suspectListScore	SuspectListScore	
retentionTimeScore	RetentionTimeScore	
CPDATCount	CPDAT_COUNT	comptox
TOXCASTActive	TOXCAST_PERCENT_ACTIVE	comptox
dataSources	DATA_SOURCES	comptox
pubChemDataSources	PUBCHEM_DATA_SOURCES	comptox
EXPOCASTPredExpo	EXPOCAST_MEDIAN_EXPOSURE_PREDICTION_MG/KG-BW/DAY	comptox
ECOTOX	ECOTOX	comptox
NORMANSUSDAT	NORMANSUSDAT	comptox
MASSBANKEU	MASSBANKEU	comptox
TOX21SL	TOX21SL	comptox
TOXCAST	TOXCAST	comptox
KEMIMARKET	KEMIMARKET	comptox
MZCLOUD	MZCLOUD	comptox
pubMedNeuro	PubMedNeuro	comptox
CIGARETTES	CIGARETTES	comptox
INDOORCT16	INDOORCT16	comptox
SRM2585DUST	SRM2585DUST	comptox
SLTCHEMDB	SLTCHEMDB	comptox
THSMOKE	THSMOKE	comptox
ITNANTIBIOTIC	ITNANTIBIOTIC	comptox
STOFFIDENT	STOFFIDENT	comptox
KEMIMARKET_EXPO	KEMIMARKET_EXPO	comptox
KEMIMARKET_HAZ	KEMIMARKET_HAZ	comptox
REACH2017	REACH2017	comptox
KEMIWW_WDUIndex	KEMIWW_WDUIndex	comptox
KEMIWW_StpSE	KEMIWW_StpSE	comptox
KEMIWW_SEHitsOverDL	KEMIWW_SEHitsOverDL	comptox
ZINC15PHARMA	ZINC15PHARMA	comptox
PFASMASTER	PFASMASTER	comptox
peakFingerprintScore	AutomatedPeakFingerprintAnnotationScore	
lossFingerprintScore	AutomatedLossFingerprintAnnotationScore	
agroChemInfo	AgroChemInfo	pubchemlite
bioPathway	BioPathway	pubchemlite
drugMedicInfo	DrugMedicInfo	pubchemlite
foodRelated	FoodRelated	pubchemlite

pharmacoInfo	PharmacoInfo	pubchemlite
safetyInfo	SafetyInfo	pubchemlite
toxicityInfo	ToxicityInfo	pubchemlite
knownUse	KnownUse	pubchemlite
disorderDisease	DisorderDisease	pubchemlite
identification	Identification	pubchemlite
annoTypeCount	FPSum	pubchemlite
annoTypeCount	AnnoTypeCount	pubchemlite
annotHitCount	AnnotHitCount	pubchemlite

In addition, the `compoundScorings` function is also useful to programmatically generate a set of scorings to be used for ranking with `MetFrag`. For instance, the following can be given to the `scoreTypes` argument to use all default scorings for PubChem: `compoundScorings("metfrag", "pubchem", onlyDefault=TRUE)$name`.

For all `MetFrag` scoring types refer to the `Candidate Scores` section on the MetFragR homepage.

Usage of MetFrag databases

When `database="chemspider"` setting the `chemSpiderToken` argument is mandatory.

If a local database is chosen via `sdf`, `psv`, or `csv` then its file location should be set with the `LocalDatabasePath` value via the `extraOpts` argument. For example: `extraOpts = list(LocalDatabasePath = "C:/myDB.csv")`.

If `database="pubchemlite"` or `database="comptox"` and **patRoonExt** is *not* installed then the file location must be specified as above or by setting the `patRoon.path.MetFragPubChemLite`/`patRoon.path.MetFragComp` option. See the installation section in the handbook for more details.

Parallelization

generateCompoundsMetFrag uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

When local database files are used with `generateCompoundsMetFrag` (*e.g.* when `database` is set to `"pubchemlite"`, `"csv"` etc.) and '`patRoon.MP.method="future"`', then the database file must be present on all the nodes. When `pubchemlite` or `comptox` is used, the location for these databases can be configured on the host with the respective package options ('`patRoon.path.MetFragPubChemLite`' and '`patRoon.path.MetFragCompTox`') or made available by installing the **patRoonExt** package. Note that these files must *also* be present on the local host computer, even if it is not participating in computations.

References

Ruttkies C, Schymanski EL, Wolf S, Hollender J, Neumann S (2016). "MetFrag relaunched: incorporating strategies beyond in silico fragmentation." *Journal of Cheminformatics*, **8**(1). doi:10.1186/s1332101601159.

See Also

`generateCompounds` for more details and other algorithms.

generateCompoundsSIRIUS *Compound annotation with SIRIUS*

Description

Uses SIRIUS in combination with CSI:FingerID for compound annotation.

Usage

```
generateCompoundsSIRIUS(fGroups, ...)

## S4 method for signature 'featureGroups'
generateCompoundsSIRIUS(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  projectPath = NULL,
  elements = "CHNOP",
  profile = "qtof",
  formulaDatabase = NULL,
  fingerIDDatabase = "pubchem",
  noise = NULL,
  cores = NULL,
  topMost = 100,
  topMostFormulas = 5,
  login = "check",
  alwaysLogin = FALSE,
  extraOptsGeneral = NULL,
  extraOptsFormula = NULL,
  verbose = TRUE,
  splitBatches = FALSE,
  dryRun = FALSE
)

## S4 method for signature 'featureGroupsSet'
generateCompoundsSIRIUS(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  projectPath = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0,
  setAvgSpecificScores = FALSE
```

)

Arguments

fGroups	featureGroups object which should be annotated. This should be the same or a subset of the object that was used to create the specified MSPeakLists. In the case of a subset only the remaining feature groups in the subset are considered.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
MSPeakLists	A MSPeakLists object that was generated for the supplied fGroups.
relMzDev	Maximum relative deviation between the measured and candidate formula *m/z* values (in ppm). Sets the '--ppm-max' command line option.
adduct	An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+". If the featureGroups object has adduct annotations then these are used if adducts=NULL. (**sets workflow**) The adduct argument is not supported for sets workflows, since the adduct annotations will then always be used.
projectPath, dryRun	These are mainly for internal purposes. projectPath sets the output directory for the SIRIUS output (a temporary directory if NULL). If dryRun is TRUE then no computations are done and only the results from projectPath are processed. (**sets workflow**) projectPath should be a character specifying the paths for each set.
elements	Elements to be considered for formulae calculation. This will heavily affects the number of candidates! Always try to work with a minimal set by excluding elements you don't expect. The minimum/maximum number of elements can also be specified, for example: a value of "C[5]H[10-15]O" will only consider formulae with up to five carbon atoms, between ten and fifteen hydrogen atoms and any amount of oxygen atoms. Sets the '--elements' command line option.
profile	Name of the configuration profile, for example: '"qtof"', '"orbitrap"', '"fticr"'. Sets the '--profile' commandline option.
formulaDatabase	If not NULL, use a database for retrieval of formula candidates. Possible values are: '"pubchem"', '"bio"', '"kegg"', '"hmdb"'. Sets the '--database' commandline option.
fingerIDDatabase	Database specifically used for CSI:FingerID. If NULL, the value of the formulaDatabase parameter will be used or "pubchem" when that is also NULL. Sets the '--fingerid-db' option.
noise	Median intensity of the noise (NULL ignores this parameter). Sets the '--noise' commandline option.
cores	The number of cores SIRIUS will use. If NULL then the default of all cores will be used.
topMost	Only keep this number of candidates (per feature group) with highest score. Set to NULL to always keep all candidates, however, please note that this may result in significant usage of CPU/RAM resources for large numbers of candidates.

`topMostFormulas`
: Do not return more than this number of candidate formulae. Note that only compounds for these formulae will be searched. Sets the '`--candidates`' commandline option.

`login`, `alwaysLogin`
: Specifies if and how account logging of SIRIUS should be handled:

`login=FALSE`: no automatic login is performed and the active login status is not checked.

`login="check"`: aborts if no active login is present.

`login="interactive"`: interactively ask for login (using **getPass**).

`login=c(username="...", password="...")`: perform the login with the given details. For security reasons, please do not enter the details directly, but use e.g. environment variables or store/retrieve them with the **keyring** package.

if `alwaysLogin=TRUE` then a login is always performed, otherwise only if SIRIUS reports no active login.

See the SIRIUS website and **patRoon** handbook for more information.

`extraOptsGeneral`, `extraOptsFormula`
: a `character` vector with any extra commandline parameters for `SIRIUS`. For `SIRIUS` versions `<4.4` there is no distinction between general and formula options. Otherwise commandline options specified in `extraOptsGeneral` are added prior to the `formula` command, while options specified in `extraOptsFormula` are added in afterwards. See the `SIRIUS` manual for more details. Set to `NULL` to ignore.

`verbose`
: If `TRUE` then more output is shown in the terminal.

`splitBatches`
: If `TRUE` then the calculations done by `SIRIUS` will be evenly split over multiple `SIRIUS` calls (which may be run in parallel depending on the set package options). If `splitBatches=FALSE` then all feature calculations are performed from a single `SIRIUS` execution, which is often the fastest if calculations are performed on a single computer.

`setThreshold`
: (**sets workflow**) Minimum abundance for a candidate among all sets ('`0-1`'). For instance, a value of '`1`' means that the candidate needs to be present in all the set data.

`setThresholdAnn`
: (**sets workflow**) As `setThreshold`, but only taking into account the set data that contain annotations for the feature group of the candidate.

`setAvgSpecificScores`
: (**sets workflow**) If `TRUE` then set specific scorings (*e.g.* MS/MS match) are also averaged.

Details

This function uses SIRIUS to generate compound candidates. This function is called when calling `generateCompounds` with `algorithm="sirius"`.

Similar to `generateFormulasSIRIUS`, candidate formulae are generated with SIRIUS. These results are then fed to CSI:FingerID to acquire candidate structures. Candidate formulae without any assigned structure will be removed (unlike `generateFormulasSIRIUS`). This method requires the availability of MS/MS data, and feature groups without it will be ignored.

Value

A compoundsSIRIUS object.

Parallelization

generateCompoundsSIRIUS uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Note

For annotations performed with SIRIUS it is often the fastest to keep the default splitBatches=FALSE.
In this case, all SIRIUS output will be printed to the terminal (unless verbose=FALSE or 'patRoon.MP.method="future"').
Furthermore, please note that only annotations to be performed for the same adduct are grouped in a single batch execution.

References

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**(4), 299–302. doi:10.1038/s4159201903448.

Duhrkop K, Bocker S (2015). "Fragmentation Trees Reloaded." In Przytycka TM (ed.), *Research in Computational Molecular Biology*, 65–79. ISBN 978-3-319-16706-0.

Duhrkop K, Shen H, Meusel M, Rousu J, Bocker S (2015). "Searching molecular structure databases with tandem mass spectra using CSI:FingerID." *Proceedings of the National Academy of Sciences*, **112**(41), 12580–12585. doi:10.1073/pnas.1509788112.

Bocker S, Letzel MC, Liptak Z, Pervukhin A (2008). "SIRIUS: decomposing isotope patterns for metabolite identification." *Bioinformatics*, **25**(2), 218–224. doi:10.1093/bioinformatics/btn603.

See Also

generateCompounds for more details and other algorithms.

---

generateFormulas *Automatic chemical formula generation*

Description

Automatically calculate chemical formulae for all feature groups.

Usage

```
generateFormulas(fGroups, MSPeakLists, algorithm, ...)

## S4 method for signature 'featureGroups'
generateFormulas(fGroups, MSPeakLists, algorithm, ...)
```

Arguments

fGroups	featureGroups object for which formulae should be generated. This should be the same or a subset of the object that was used to create the specified MSPeakLists. In the case of a subset only the remaining feature groups in the subset are considered.
MSPeakLists	An MSPeakLists object that was generated for the supplied fGroups.
algorithm	A character string describing the algorithm that should be used: "bruker", "genform", "sirius"
...	Any parameters to be passed to the selected formula generation algorithm.

Details

Several algorithms are provided to automatically generate formulae for given feature groups. All algorithms use the accurate mass of a feature to back-calculate candidate formulae. Depending on the algorithm and data availability, other data such as isotopic pattern and MS/MS fragments may be used to further improve formula assignment and ranking.

generateFormulas is a generic function that will generateFormulas by one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as generateFormulasDA and generateFormulasGenForm. While these functions may be called directly, generateFormulas provides a generic interface and is therefore usually preferred.

Value

A formulas object containing all generated formulae.

Candidate assignment

Formula candidate assignment occurs in one of the following ways:

- Candidates are first generated for each feature and then pooled to form consensus candidates for the feature group.
- Candidates are directly generated for each feature group by group averaged MS peak list data.

With approach (1), scorings and mass errors are averaged and outliers are removed (controlled by featThreshold and featThresholdAnn arguments). Other candidate properties that cannot be averaged are from the feature from the analysis as specified in the "analysis" column of the results. The second approach only generates candidate formulae once for every feature group, and is therefore generally much faster. However, this inherently prevents removal of outliers.

Note that with either approach subsequent workflow steps that use formula data (*e.g.* addFormulaScoring and reporting functions) only use formula data that was eventually assigned to feature groups.

Scorings

Each algorithm implements their own scoring system. Their names have been harmonized where possible. An overview is obtained with the formulaScorings function:

name	genform	sirius	bruker	description

combMatch	comb_match	-	-	MS and MS/MS combined match value
isoScore	MS_match	isoScore	-	How well the isotopic pattern matches
mSigma	-	-	mSigma	Deviation of the isotopic pattern
MSMSScore	MSMS_match	treeScore	-	How well MS/MS data matches
score	-	score	Score	Overall MS formula score

Sets workflows

With a sets workflow, annotation is first performed for each set. This is important, since the annotation algorithms typically cannot work with data from mixed ionization modes. The annotation results are then combined to generate a *sets consensus*:

- The annotation tables for each feature group from the set specific data are combined. Rows with overlapping candidates (determined by the neutral formula) are merged.
- Set specific data (*e.g.* the ionic formula) is retained by renaming their columns with set specific names.
- The MS/MS fragment annotations (`fragInfo` column) from each set are combined.
- The scorings for each set are averaged to calculate overall scores. if `setAvgSpecificScores=FALSE` then scorings that are considered set specific (*e.g.* MS/MS and isotopic pattern match) are *not* averaged.
- The candidates are re-ranked based on their average ranking among the set data (if a candidate is absent in a set it is assigned the poorest rank in that set).
- The coverage of each candidate among sets is calculated. Depending on the `setThreshold` and `setThresholdAnn` arguments, candidates with low abundance are removed.

See Also

The `formulas` output class and its methods and the algorithm specific functions: `generateFormulasDA`, `generateFormulasGenForm`, `generateFormulasSIRIUS`

The GenForm manual (also known as MOLGEN-MSMS).

`generateFormulasDA` *Generate formula with Bruker DataAnalysis*

Description

Uses Bruker DataAnalysis to generate chemical formulae.

Usage

```
generateFormulasDA(fGroups, ...)

## S4 method for signature 'featureGroups'
generateFormulasDA(
  fGroups,
  MSPeakLists,
```

```
  precursorMzSearchWindow = 0.002,
  MSMode = "both",
  adduct = NULL,
  featThreshold = 0,
  featThresholdAnn = 0.75,
  absAlignMzDev = 0.002,
  save = TRUE,
  close = save
)

## S4 method for signature 'featureGroupsSet'
generateFormulasDA(
  fGroups,
  MSPeakLists,
  precursorMzSearchWindow = 0.002,
  MSMode = "both",
  adduct = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0,
  setAvgSpecificScores = FALSE
)
```

Arguments

fGroups	featureGroups object for which formulae should be generated. This should be the same or a subset of the object that was used to create the specified MSPeakLists. In the case of a subset only the remaining feature groups in the subset are considered.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
MSPeakLists	An MSPeakLists object that was generated for the supplied fGroups.
precursorMzSearchWindow	Search window for *m/z* values (+/- the feature *m/z*) used to find back feature data of precursor/parent ions from MS/MS spectra (this data is not readily available from SmartFormula3D results).
MSMode	Whether formulae should be generated only from MS data ("ms"), MS/MS data ("msms") or both ("both"). Selecting "both" will calculate formulae from MS data and MS/MS data and combines the results (duplicated formulae are removed). This is useful when poor MS/MS data would exclude proper candidates.
adduct	An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+". If the featureGroups object has adduct annotations then these are used if adducts=NULL. (**sets workflow**) The adduct argument is not supported for sets workflows, since the adduct annotations will then always be used.
featThreshold	If calculateFeatures=TRUE: minimum presence ('0-1') of a formula in all features before it is considered as a candidate for a feature group. For instance,

featThreshold=0.75 dictates that a formula should be present in at least 75% of the features inside a feature group.

featThresholdAnn
: As featThreshold, but only considers features with annotations. For instance, featThresholdAnn=0.75 dictates that a formula should be present in at least 75% of the features with annotations inside a feature group.

absAlignMzDev
: When the group formula annotation consensus is made from feature annotations, the *m/z* values of annotated MS/MS fragments may slightly deviate from those of the corresponding group MS/MS peak list. The absAlignMzDev argument specifies the maximum *m/z* window used to re-align the mass peaks.

close, save
: If TRUE then Bruker files are closed and saved after processing with DataAnalysis, respectively. Setting close=TRUE prevents that many analyses might be opened simultaneously in DataAnalysis, which otherwise may use excessive memory or become slow. By default save is TRUE when close is TRUE, which is likely what you want as otherwise any processed data is lost.

setThreshold
: (**sets workflow**) Minimum abundance for a candidate among all sets ('0-1'). For instance, a value of '1' means that the candidate needs to be present in all the set data.

setThresholdAnn
: (**sets workflow**) As setThreshold, but only taking into account the set data that contain annotations for the feature group of the candidate.

setAvgSpecificScores
: (**sets workflow**) If TRUE then set specific scorings (*e.g.* MS/MS match) are also averaged.

Details

This function uses bruker to generate formula candidates. This function is called when calling generateFormulas with algorithm="bruker".

This method supports scoring based on overlap between measured and theoretical isotopic patterns (both MS and MS/MS data) and the presence of 'fitting' MS/MS fragments. The method will iterate through all features (or "Compounds" in DataAnalysis terms) and call SmartFormula (and SmartFormula3D if MS/MS data is available) to generate all formulae. Parameters affecting formula calculation have to be set in advance within the DataAnalysis method for each analysis (*e.g.* by setDAMethod).

This method requires that features were obtained with findFeaturesBruker. It is recommended, but not mandatory, that the MSPeakLists are also generated by DataAnalysis.

Calculation of formulae with DataAnalysis always occurs with the 'feature approach' (see Candidate assignment in generateFormulas).

Value

A formulas object containing all generated formulae.

Note

If any errors related to DCOM appear it might be necessary to terminate DataAnalysis (note that DataAnalysis might still be running as a background process). The ProcessCleaner application installed with DataAnalayis can be used for this.

See Also

generateFormulas for more details and other algorithms.

---

generateFormulasGenForm

Generate formula with GenForm

Description

Uses GenForm to generate chemical formula candidates.

Usage

```
generateFormulasGenForm(fGroups, ...)

## S4 method for signature 'featureGroups'
generateFormulasGenForm(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  elements = "CHNOP",
  hetero = TRUE,
  oc = FALSE,
  thrMS = NULL,
  thrMSMS = NULL,
  thrComb = NULL,
  maxCandidates = Inf,
  extraOpts = NULL,
  calculateFeatures = TRUE,
  featThreshold = 0,
  featThresholdAnn = 0.75,
  absAlignMzDev = 0.002,
  MSMode = "both",
  isolatePrec = TRUE,
  timeout = 120,
  topMost = 50,
  batchSize = 8
)

## S4 method for signature 'featureGroupsSet'
```

```
generateFormulasGenForm(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0,
  setAvgSpecificScores = FALSE
)
```

Arguments

fGroups	featureGroups object for which formulae should be generated. This should be the same or a subset of the object that was used to create the specified MSPeakLists. In the case of a subset only the remaining feature groups in the subset are considered.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
MSPeakLists	An MSPeakLists object that was generated for the supplied fGroups.
relMzDev	Maximum relative deviation between the measured and candidate formula *m/z* values (in ppm). Sets the 'ppm' command line option.
adduct	An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+". If the featureGroups object has adduct annotations then these are used if adducts=NULL. (**sets workflow**) The adduct argument is not supported for sets workflows, since the adduct annotations will then always be used.
elements	Elements to be considered for formulae calculation. This will heavily affects the number of candidates! Always try to work with a minimal set by excluding elements you don't expect. Sets the 'el' command line option.
hetero	Only consider formulae with at least one hetero atom. Sets the 'het' command-line option.
oc	Only consider organic formulae (*i.e.* with at least one carbon atom). Sets the 'oc' commandline option.
thrMS, thrMSMS, thrComb	Sets the thresholds for the GenForm MS score (isoScore), MS/MS score (MSMSScore) and combined score (combMatch). Sets the 'thms'/'thmsms'/'thcomb' command line options, respectively. Set to NULL for no threshold.
maxCandidates	If this number of candidates are found then GenForm aborts any further formula calculations. The number of candidates is determined *after* any formula filters, hence, the properties and 'quality' of the candidates is influenced by options such as oc and thrMS arguments. Note that this is different than topMost, which selects the candidates after GenForm finished. Sets the 'max' command line option. Set to '0' or Inf for no maximum.
extraOpts	An optional character vector with any other command line options that will be passed to GenForm. See the GenForm options section for all available command line options.

calculateFeatures
: If TRUE fomulae are first calculated for all features prior to feature group assignment (see Candidate assignment in generateFormulas).

featThreshold
: If calculateFeatures=TRUE: minimum presence ('0-1') of a formula in all features before it is considered as a candidate for a feature group. For instance, featThreshold=0.75 dictates that a formula should be present in at least 75% of the features inside a feature group.

featThresholdAnn
: As featThreshold, but only considers features with annotations. For instance, featThresholdAnn=0.75 dictates that a formula should be present in at least 75% of the features with annotations inside a feature group. @param topMost Only keep this number of candidates (per feature group) with highest score.

absAlignMzDev
: When the group formula annotation consensus is made from feature annotations, the *m/z* values of annotated MS/MS fragments may slightly deviate from those of the corresponding group MS/MS peak list. The absAlignMzDev argument specifies the maximum *m/z* window used to re-align the mass peaks.

MSMode
: Whether formulae should be generated only from MS data ("ms"), MS/MS data ("msms") or both ("both"). Selecting "both" will fall back to formula calculation with only MS data in case no MS/MS data is available.

isolatePrec
: Settings used for isolation of precursor mass peaks and their isotopes. This isolation is highly important for accurate isotope scoring of candidates, as non-relevant mass peaks will dramatically decrease the score. The value of isolatePrec should either be a list with parameters (see the filter method for MSPeakLists for more details), TRUE for default parameters or FALSE for no isolation (*e.g.* when you already performed isolation with the filter method). The z parameter (charge) is automatically deduced from the adduct used for annotation (unless isolatePrec=FALSE), hence any custom z setting is ignored.

timeout
: Maximum time (in seconds) that a GenForm command is allowed to execute. If this time is exceeded a warning is emitted and the command is terminated. See the notes section for more information on the need of timeouts.

topMost
: Only keep this number of candidates (per feature group) with highest score.

batchSize
: Maximum number of GenForm commands that should be run sequentially in each parallel process. Combining commands with short runtimes (such as GenForm) can significantly increase parallel performance. For more information see executeMultiProcess. Note that this is ignored if 'patRoon.MP.method="future"'.

setThreshold
: (**sets workflow**) Minimum abundance for a candidate among all sets ('0-1'). For instance, a value of '1' means that the candidate needs to be present in all the set data.

setThresholdAnn
: (**sets workflow**) As setThreshold, but only taking into account the set data that contain annotations for the feature group of the candidate.

setAvgSpecificScores
: (**sets workflow**) If TRUE then set specific scorings (*e.g.* MS/MS match) are also averaged.

Details

This function uses genform to generate formula candidates. This function is called when calling `generateFormulas` with `algorithm="genform"`.

When MS/MS data is available it will be used to score candidate formulae by presence of 'fitting' fragments.

Value

A `formulas` object containing all generated formulae.

GenForm options

Below is a list of options (generated by running `GenForm` without commandline options) which can be set by the `extraOpts` parameter.

```
Formula calculation from MS and MS/MS data as described in
Meringer et al (2011) MATCH Commun Math Comput Chem 65: 259-290
Usage: GenForm ms=<filename> [msms=<filename>] [out=<filename>]
        [exist[=mv]] [m=<number>] [ion=-e|+e|-H|+H|+Na] [cha=<number>]
        [ppm=<number>] [msmv=ndp|nsse|nsae] [acc=<number>] [rej=<number>]
        [thms=<number>] [thmsms=<number>] [thcomb=<number>]
      [sort[=ppm|msmv|msmsmv|combmv]] [el=<elements> [oc]] [ff=<fuzzy formula>]
     [vsp[=<even|odd>]] [vsm2mv[=<value>]] [vsm2ap2[=<value>]] [hcf] [kfer[=ex]]
        [wm[=lin|sqrt|log]] [wi[=lin|sqrt|log]] [exp=<number>] [oei]
        [dbeexc=<number>] [ivsm2mv=<number>] [vsm2ap2=<number>]
        [oms[=<filename>]] [omsms[=<filename>]] [oclean[=<filename>]]
        [analyze [loss] [intens]] [dbe] [cm] [pc] [sc] [max]
Explanation:
        ms      : filename of MS data (*.txt)
        msms    : filename of MS/MS data (*.txt)
        out     : output generated formulas
        exist   : allow only molecular formulas for that at least one
                  structural formula exists;overrides vsp, vsm2mv, vsm2ap2;
                  argument mv enables multiple valencies for P and S
        m       : experimental molecular mass (default: mass of MS basepeak)
        ion     : type of ion measured (default: M+H)
        ppm     : accuracy of measurement in parts per million (default: 5)
        msmv    : MS match value based on normalized dot product, normalized
                  sum of squared or absolute errors (default: nsae)
        acc     : allowed deviation for full acceptance of MS/MS peak in ppm
                  (default: 2)
        rej     : allowed deviation for total rejection of MS/MS peak in ppm
                  (default: 4)
        thms    : threshold for the MS match value
        thmsms  : threshold for the MS/MS match value
        thcomb  : threshold for the combined match value
        sort    : sort generated formulas according to mass deviation in ppm,
                  MS match value, MS/MS match value or combined match value
```

```
el      : used chemical elements (default: CHBrClFINOPSSi)
oc      : only organic compounds, i.e. with at least one C atom
ff      : overwrites el and oc and uses fuzzy formula for limits of
          element multiplicities
het     : formulas must have at least one hetero atom
vsp     : valency sum parity (even for graphical formulas)
vsm2mv  : lower bound for valency sum - 2 * maximum valency
          (>=0 for graphical formulas)
vsm2ap2 : lower bound for valency sum - 2 * number of atoms + 2
          (>=0 for graphical connected formulas)
hcf     : apply Heuerding-Clerc filter
kfer    : apply Kind-Fiehn element ratio (extended) ranges
wm      : m/z weighting for MS/MS match value
wi      : intensity weighting for MS/MS match value
exp     : exponent used, when wi is set to log
oei     : allow odd electron ions for explaining MS/MS peaks
dbeexc  : excess of double bond equivalent for ions
ivsm2mv : lower bound for valency sum - 2 * maximum valency
          for fragment ions
ivsm2ap2: lower bound for valency sum - 2 * number of atoms + 2
          for fragment ions
oms     : write scaled MS peaks to output
omsms   : write weighted MS/MS peaks to output
oclean  : write explained MS/MS peaks to output
analyze : write explanations for MS/MS peaks to output
loss    : for analyzing MS/MS peaks write losses instead of fragments
intens  : write intensities of MS/MS peaks to output
dbe     : write double bond equivalents to output
cm      : write calculated ion masses to output
pc      : output match values in percent
sc      : strip calculated isotope distributions
noref   : hide the reference information
max     : maximum number of final candidates (0 is no limit)
```

Parallelization

generateFormulasGenForm uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

When `futures` are used for parallel processing (`patRoon.MP.method="future"`), calculations with `GenForm` are done with batch mode disabled (see `batchSize` argument), which generally limit overall performance.

Note

This function always sets the '`exist`' and '`oei`' `GenForm` command line options.

Formula calculation with `GenForm` may produce an excessive number of candidates for high *m/z* values (*e.g.* above 600) and/or many elemental combinations (set by `elements`). In this scenario

formula calculation may need a very long time. Timeouts are used to avoid excessive computational times by terminating long running commands (set by the `timeout` argument).

References

Meringer M, Reinker S, Zhang J, Muller A (2011). "MS/MS Data Improves Automated Determination of Molecular Formulas by Mass Spectrometry." *MATCH Commun. Math. Comput. Chem.*, **65**(2), 259–290.

See Also

`generateFormulas` for more details and other algorithms.

`generateFormulasSIRIUS` *Generate formula with SIRIUS*

Description

Uses SIRIUS to generate chemical formulae candidates.

Usage

```
generateFormulasSIRIUS(fGroups, ...)

## S4 method for signature 'featureGroups'
generateFormulasSIRIUS(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  projectPath = NULL,
  elements = "CHNOP",
  profile = "qtof",
  database = NULL,
  noise = NULL,
  cores = NULL,
  getFingerprints = FALSE,
  topMost = 100,
  login = FALSE,
  alwaysLogin = FALSE,
  extraOptsGeneral = NULL,
  extraOptsFormula = NULL,
  calculateFeatures = TRUE,
  featThreshold = 0,
  featThresholdAnn = 0.75,
  absAlignMzDev = 0.002,
```

```
  verbose = TRUE,
  splitBatches = FALSE,
  dryRun = FALSE
)

## S4 method for signature 'featureGroupsSet'
generateFormulasSIRIUS(
  fGroups,
  MSPeakLists,
  relMzDev = 5,
  adduct = NULL,
  projectPath = NULL,
  ...,
  setThreshold = 0,
  setThresholdAnn = 0,
  setAvgSpecificScores = FALSE
)
```

Arguments

fGroups	featureGroups object for which formulae should be generated. This should be the same or a subset of the object that was used to create the specified MSPeakLists. In the case of a subset only the remaining feature groups in the subset are considered.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
MSPeakLists	An MSPeakLists object that was generated for the supplied fGroups.
relMzDev	Maximum relative deviation between the measured and candidate formula *m/z* values (in ppm). Sets the '--ppm-max' command line option.
adduct	An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+". If the featureGroups object has adduct annotations then these are used if adducts=NULL. (**sets workflow**) The adduct argument is not supported for sets workflows, since the adduct annotations will then always be used.
projectPath, dryRun	These are mainly for internal purposes. projectPath sets the output directory for the SIRIUS output (a temporary directory if NULL). If dryRun is TRUE then no computations are done and only the results from projectPath are processed. (**sets workflow**) projectPath should be a character specifying the paths for each set.
elements	Elements to be considered for formulae calculation. This will heavily affects the number of candidates! Always try to work with a minimal set by excluding elements you don't expect. The minimum/maximum number of elements can also be specified, for example: a value of "C[5]H[10-15]O" will only consider formulae with up to five carbon atoms, between ten and fifteen hydrogen atoms and any amount of oxygen atoms. Sets the '--elements' command line option.
profile	Name of the configuration profile, for example: '"qtof"', '"orbitrap"', '"fticr"'. Sets the '--profile' commandline option.

`database` If not `NULL`, use a database for retrieval of formula candidates. Possible values are: '`"pubchem"`', '`"bio"`', '`"kegg"`', '`"hmdb"`'. Sets the '`--database`' commandline option.

`noise` Median intensity of the noise (`NULL` ignores this parameter). Sets the '`--noise`' commandline option.

`cores` The number of cores `SIRIUS` will use. If `NULL` then the default of all cores will be used.

`getFingerprints`
Set to `TRUE` to load `SIRIUS-CSI:FingerID` MS/MS fingerprints for the formula candidates. This is currently only supported with `calculateFeatures=FALSE` to avoid heavy server traffic. The fingerprints are stored in the `fingerprints` slot of the returned `formulasSIRIUS` object, and are used by the `predictTox` and `predictRespFactors` methods.

`topMost` Only keep this number of candidates (per feature group) with highest score. Sets the '`--candidates`' command line option.

`login`, `alwaysLogin`
Specifies if and how account logging of SIRIUS should be handled:
`login=FALSE`: no automatic login is performed and the active login status is not checked.
`login="check"`: aborts if no active login is present.
`login="interactive"`: interactively ask for login (using **getPass**).
`login=c(username="...", password="...")`: perform the login with the given details. For security reasons, please do not enter the details directly, but use e.g. environment variables or store/retrieve them with the **keyring** package.
if `alwaysLogin=TRUE` then a login is always performed, otherwise only if SIRIUS reports no active login.
See the SIRIUS website and **patRoon** handbook for more information.

`extraOptsGeneral`, `extraOptsFormula`
a `character` vector with any extra commandline parameters for `SIRIUS`. For `SIRIUS` versions `<4.4` there is no distinction between general and formula options. Otherwise commandline options specified in `extraOptsGeneral` are added prior to the `formula` command, while options specified in `extraOptsFormula` are added in afterwards. See the `SIRIUS` manual for more details. Set to `NULL` to ignore.

`calculateFeatures`
If `TRUE` fomulae are first calculated for all features prior to feature group assignment (see `Candidate assignment` in `generateFormulas`).

`featThreshold` If `calculateFeatures=TRUE`: minimum presence ('`0-1`') of a formula in all features before it is considered as a candidate for a feature group. For instance, `featThreshold=0.75` dictates that a formula should be present in at least 75% of the features inside a feature group.

`featThresholdAnn`
As `featThreshold`, but only considers features with annotations. For instance, `featThresholdAnn=0.75` dictates that a formula should be present in at least 75% of the features with annotations inside a feature group. @param topMost Only keep this number of candidates (per feature group) with highest score. Sets the '`--candidates`' command line option.

`absAlignMzDev` When the group formula annotation consensus is made from feature annotations, the *m/z* values of annotated MS/MS fragments may slightly deviate from those of the corresponding group MS/MS peak list. The `absAlignMzDev` argument specifies the maximum *m/z* window used to re-align the mass peaks.

`verbose` If `TRUE` then more output is shown in the terminal.

`splitBatches` If `TRUE` then the calculations done by `SIRIUS` will be evenly split over multiple `SIRIUS` calls (which may be run in parallel depending on the set package options). If `splitBatches=FALSE` then all feature calculations are performed from a single `SIRIUS` execution, which is often the fastest if calculations are performed on a single computer.

`setThreshold` (**sets workflow**) Minimum abundance for a candidate among all sets (‘`0-1`’). For instance, a value of ‘`1`’ means that the candidate needs to be present in all the set data.

`setThresholdAnn` (**sets workflow**) As `setThreshold`, but only taking into account the set data that contain annotations for the feature group of the candidate.

`setAvgSpecificScores` (**sets workflow**) If `TRUE` then set specific scorings (*e.g.* MS/MS match) are also averaged.

Details

This function uses sirius to generate formula candidates. This function is called when calling `generateFormulas` with `algorithm="sirius"`.

Similarity of measured and theoretical isotopic patterns will be used for scoring candidates. Note that `SIRIUS` requires availability of MS/MS data.

Value

A `formulasSIRIUS` object.

Parallelization

generateFormulasSIRIUS uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Note

For annotations performed with `SIRIUS` it is often the fastest to keep the default `splitBatches=FALSE`.
In this case, all `SIRIUS` output will be printed to the terminal (unless `verbose=FALSE` or ‘`patRoon.MP.method="future"`’).
Furthermore, please note that only annotations to be performed for the same adduct are grouped in a single batch execution.

References

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019). “SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information.” *Nature Methods*, **16**(4), 299–302. doi:10.1038/s4159201903448.

Duhrkop K, Bocker S (2015). "Fragmentation Trees Reloaded." In Przytycka TM (ed.), *Research in Computational Molecular Biology*, 65–79. ISBN 978-3-319-16706-0.

Duhrkop K, Shen H, Meusel M, Rousu J, Bocker S (2015). "Searching molecular structure databases with tandem mass spectra using CSI:FingerID." *Proceedings of the National Academy of Sciences*, **112**(41), 12580–12585. doi:10.1073/pnas.1509788112.

Bocker S, Letzel MC, Liptak Z, Pervukhin A (2008). "SIRIUS: decomposing isotope patterns for metabolite identification." *Bioinformatics*, **25**(2), 218–224. doi:10.1093/bioinformatics/btn603.

See Also

`generateFormulas` for more details and other algorithms.

`generateMSPeakLists` *Generation of MS Peak Lists*

Description

Functionality to convert MS and MS/MS data into MS peak lists.

Usage

```
generateMSPeakLists(fGroups, algorithm, ...)

## S4 method for signature 'featureGroups'
generateMSPeakLists(fGroups, algorithm, ...)
```

Arguments

`fGroups`	The `featureGroups` object from which MS peak lists should be extracted.
`algorithm`	A character string describing the algorithm that should be used: `"bruker"`, `"brukerfmf"`, `"mzr"`
`...`	Any parameters to be passed to the selected MS peak lists generation algorithm.

Details

Formula calculation and identification tools rely on mass spectra that belong to features of interest. For processing, MS (and MS/MS) spectra are typically reduced to a table with a column containing measured *m/z* values and a column containing their intensities. These 'MS peak lists' can then be used for formula generation and compound generation.

MS and MS/MS peak lists are first generated for all features (or a subset, if the `topMost` argument is set). During this step multiple spectra over the feature elution profile are averaged. Subsequently, peak lists will be generated for each feature group by averaging peak lists of the features within the group. Functionality that uses peak lists will either use data from individual features or from group

averaged peak lists. For instance, the former may be used by formulae calculation, while compound identification and plotting functionality typically uses group averaged peak lists.

`generateMSPeakLists` is a generic function that will generateMSPeakLists by one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as `generateMSPeakListsMzR` and `generateMSPeakListsDA`. While these functions may be called directly, `generateMSPeakLists` provides a generic interface and is therefore usually preferred.

Value

A `MSPeakLists` object.

Sets workflows

With a sets workflow, the feature group averaged peak lists are made per set. This is important, because for averaging peak lists cannot be mixed, for instance, when different ionization modes were used to generate the sets. The group averaged peaklists are then simply combined and labelled in the final peak lists. However, please note that annotation and other functionality typically uses only the set specific peak lists, as this functionality cannot work with mixed peak lists.

Note

In most cases it will be necessary to centroid your MS input files. The only exception is `Bruker`, however, you will still need centroided '`mzXML`'/'`mzML`' files for *e.g.* plotting chromatograms. In this case the centroided MS files should be stored in the same directory as the raw `Bruker` '`.d`' files. The `convertMSFiles` function can be used to centroid data.

See Also

The `MSPeakLists` output class and its methods and the algorithm specific functions: `generateMSPeakListsDA`, `generateMSPeakListsDAFMF`, `generateMSPeakListsMzR`

---

generateMSPeakListsDA *Generate peak lists with Bruker DataAnalysis*

Description

Uses Bruker DataAnalysis to read the data needed to generate MS peak lists.

Usage

```
generateMSPeakListsDA(fGroups, ...)

## S4 method for signature 'featureGroups'
generateMSPeakListsDA(
  fGroups,
  bgsubtr = TRUE,
  maxMSRtWindow = 5,
```

```
  minMSIntensity = 500,
  minMSMSIntensity = 500,
  clear = TRUE,
  close = TRUE,
  save = close,
  MSMSType = "MSMS",
  avgFGroupParams = getDefAvgPListParams()
)

## S4 method for signature 'featureGroupsSet'
generateMSPeakListsDA(fGroups, ...)
```

Arguments

fGroups	The featureGroups object from which MS peak lists should be extracted.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
bgsubtr	If TRUE background will be subtracted using the 'spectral' algorithm.
maxMSRtWindow	Maximum chromatographic peak window used for spectrum averaging (in seconds, +/- retention time). If NULL all spectra from a feature will be taken into account. Lower to decrease processing time.
minMSIntensity, minMSMSIntensity	Minimum intensity for peak lists obtained with DataAnalysis. Highly recommended to set '>0' as DA tends to report many very low intensity peaks.
clear	Remove any existing chromatogram traces/mass spectra prior to making new ones.
close, save	If TRUE then Bruker files are closed and saved after processing with DataAnalysis, respectively. Setting close=TRUE prevents that many analyses might be opened simultaneously in DataAnalysis, which otherwise may use excessive memory or become slow. By default save is TRUE when close is TRUE, which is likely what you want as otherwise any processed data is lost.
MSMSType	The type of MS/MS experiment performed: "MSMS" for MRM/AutoMSMS or "BBCID" for broadband CID.
avgFGroupParams	A list with parameters used for averaging of peak lists for feature groups. See getDefAvgPListParams for more details.

Details

This function uses Bruker DataAnalysis to generate MS peak lists. This function is called when calling generateMSPeakLists with algorithm="bruker".

The MS data should be in the Bruker data format ('.d'). This function leverages DataAnalysis functionality to support averaging of spectra, background subtraction and identification of isotopes. In order to obtain mass spectra TICs will be added in DataAnalysis of the MS and relevant MS/MS signals.

Value

A MSPeakLists object.

Note

The 'Component' column should be active (Method–>Parameters–>Layouts–>Mass List Layout) in order to add isotopologue information.

If any errors related to DCOM appear it might be necessary to terminate DataAnalysis (note that DataAnalysis might still be running as a background process). The ProcessCleaner application installed with DataAnalayis can be used for this.

See Also

generateMSPeakLists for more details and other algorithms.

---

generateMSPeakListsDAFMF

Generate peak lists with Bruker DataAnalysis from bruker features

---

Description

Uses 'compounds' that were generated by the Find Molecular Features (FMF) algorithm of Bruker DataAnalysis to extract MS peak lists.

Usage

```
generateMSPeakListsDAFMF(fGroups, ...)

## S4 method for signature 'featureGroups'
generateMSPeakListsDAFMF(
  fGroups,
  minMSIntensity = 500,
  minMSMSIntensity = 500,
  close = TRUE,
  save = close,
  avgFGroupParams = getDefAvgPListParams()
)

## S4 method for signature 'featureGroupsSet'
generateMSPeakListsDAFMF(fGroups, ...)
```

Arguments

fGroups	The featureGroups object from which MS peak lists should be extracted.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
minMSIntensity, minMSMSIntensity	Minimum intensity for peak lists obtained with DataAnalysis. Highly recommended to set '>0' as DA tends to report many very low intensity peaks.

close, save — If TRUE then Bruker files are closed and saved after processing with DataAnalysis, respectively. Setting close=TRUE prevents that many analyses might be opened simultaneously in DataAnalysis, which otherwise may use excessive memory or become slow. By default save is TRUE when close is TRUE, which is likely what you want as otherwise any processed data is lost.

avgFGroupParams — A list with parameters used for averaging of peak lists for feature groups. See getDefAvgPListParams for more details.

Details

This function uses Bruker DataAnalysis with FMF to generate MS peak lists. This function is called when calling generateMSPeakLists with algorithm="brukerfmf".

This function is similar to generateMSPeakListsDA, but uses 'compounds' that were generated by the Find Molecular Features (FMF) algorithm to extract MS peak lists. This is generally much faster , however, it only works when features were obtained with the findFeaturesBruker function. Since all MS spectra are generated in advance by Bruker DataAnalysis, only few parameters exist to customize its operation.

Value

A MSPeakLists object.

Note

If any errors related to DCOM appear it might be necessary to terminate DataAnalysis (note that DataAnalysis might still be running as a background process). The ProcessCleaner application installed with DataAnalayis can be used for this.

See Also

generateMSPeakLists for more details and other algorithms.

---

generateMSPeakListsMzR

Generate peak lists with mzR

Description

Uses the **mzR** package to read the MS data needed for MS peak lists.

Usage

```
generateMSPeakListsMzR(fGroups, ...)

## S4 method for signature 'featureGroups'
generateMSPeakListsMzR(
  fGroups,
  maxMSRtWindow = 5,
  precursorMzWindow = 4,
  topMost = NULL,
  avgFeatParams = getDefAvgPListParams(),
  avgFGroupParams = getDefAvgPListParams()
)

## S4 method for signature 'featureGroupsSet'
generateMSPeakListsMzR(fGroups, ...)
```

Arguments

fGroups	The featureGroups object from which MS peak lists should be extracted.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
maxMSRtWindow	Maximum chromatographic peak window used for spectrum averaging (in seconds, +/- retention time). If NULL all spectra from a feature will be taken into account. Lower to decrease processing time.
precursorMzWindow	The *m/z* window (in Da) to find MS/MS spectra of a precursor. This is typically used for Data-Dependent like MS/MS data and should correspond to the isolation *m/z* window (*i.e.* +/- the precursor *m/z*) that was used to collect the data. For Data-Independent MS/MS experiments, where precursor ions are not isolated prior to fragmentation (*e.g.* bbCID, MSe, all-ion, ...) the value should be NULL.
topMost	Only extract MS peak lists from a maximum of topMost analyses with highest intensity. If NULL all analyses will be used.
avgFeatParams	Parameters used for averaging MS peak lists of individual features. Analogous to avgFGroupParams.
avgFGroupParams	A list with parameters used for averaging of peak lists for feature groups. See getDefAvgPListParams for more details.

Details

This function uses mzR to generate MS peak lists. This function is called when calling generateMSPeakLists with algorithm="mzr".

The MS data files should be either in '.mzXML' or '.mzML' format.

The input MS data files need to be centroided. The convertMSFiles function can be used to centroid data.

Value

A MSPeakLists object.

References

Chambers, C. M, Maclean, Brendan, Burke, Robert, Amodei, Dario, Ruderman, L. D, Neumann, Steffen, Gatto, Laurent, Fischer, Bernd, Pratt, Brian, Egertson, Jarrett, Hoff, Katherine, Kessner, Darren, Tasman, Natalie, Shulman, Nicholas, Frewen, Barbara, Baker, A. T, Brusniak, Mi-Youn, Paulse, Christopher, Creasy, David, Flashner, Lisa, Kani, Kian, Moulding, Chris, Seymour, L. S, Nuwaysir, M. L, Lefebvre, Brent, Kuhlmann, Frank, Roark, Joe, Rainer, Paape, Detlev, Suckau, Hemenway, Tina, Huhmer, Andreas, Langridge, James, Connolly, Brian, Chadick, Trey, Holly, Krisztina, Eckels, Josh, Deutsch, W. E, Moritz, L. R, Katz, E. J, Agus, B. D, MacCoss, Michael, Tabb, L. D, Mallick, Parag (2012). "A cross-platform toolkit for mass spectrometry and proteomics." *Nat Biotech*, **30**(10), 918–920. doi:10.1038/nbt.2377, http://dx.doi.org/10.1038/nbt.2377.

Keller A, Eng J, Zhang N, Li X, Aebersold R (2005). "A uniform proteomics MS/MS analysis platform utilizing open XML file formats." *Mol Syst Biol*.

Kessner D, Chambers M, Burke R, Agus D, Mallick P (2008). "ProteoWizard: open source software for rapid proteomics tools development." *Bioinformatics*, **24**(21), 2534–2536. doi:10.1093/bioinformatics/btn323.

Martens L, Chambers M, Sturm M, Kessner D, Levander F, Shofstahl J, Tang WH, Rompp A, Neumann S, Pizarro AD, Montecchi-Palazzi L, Tasman N, Coleman M, Reisinger F, Souda P, Hermjakob H, Binz P, Deutsch EW (2010). "mzML - a Community Standard for Mass Spectrometry Data." *Mol Cell Proteomics*. doi:10.1074/mcp.R110.000133.

Pedrioli PGA, Eng JK, Hubley R, Vogelzang M, Deutsch EW, Raught B, Pratt B, Nilsson E, Angeletti RH, Apweiler R, Cheung K, Costello CE, Hermjakob H, Huang S, Julian RK, Kapp E, McComb ME, Oliver SG, Omenn G, Paton NW, Simpson R, Smith R, Taylor CF, Zhu W, Aebersold R (2004). "A common open representation of mass spectrometry data and its application to proteomics research." *Nat Biotechnol*, **22**(11), 1459–1466. doi:10.1038/nbt1031.

See Also

generateMSPeakLists for more details and other algorithms.

---

generateTPs *Generation of transformation products (TPs)*

Description

Functionality to automatically obtain transformation products for a given set of parent compounds.

Usage

```
generateTPs(algorithm, ...)
```

Arguments

algorithm	A character string describing the algorithm that should be used: "biotransformer", "logic", "library", "library_formula", "cts"
...	Any parameters to be passed to the selected TP generation algorithm.

Details

generateTPs is a generic function that will generate transformation products by one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as generateTPsBioTransformer and generateTPsLogic. While these functions may be called directly, generateTPs provides a generic interface and is therefore usually preferred.

Value

A transformationProducts (derived) object containing all generated TPs.

See Also

The transformationProducts output class and its methods and the algorithm specific functions: generateTPsBioTransformer, generateTPsLogic, generateTPsLibrary, generateTPsLibraryFormula, generateTPsCTS

The derived class transformationProductsStructure for more specific methods to post-process TP data.

---

generateTPsBioTransformer

Obtain transformation products (TPs) with BioTransformer

Description

Uses BioTransformer to predict TPs

Usage

```
generateTPsBioTransformer(
  parents,
  type = "env",
  generations = 2,
  maxExpGenerations = generations + 2,
  extraOpts = NULL,
  skipInvalid = TRUE,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE,
  neutralizeTPs = TRUE,
  calcSims = FALSE,
  fpType = "extended",
```

```
  fpSimMethod = "tanimoto",
  MP = FALSE
)
```

Arguments

parents	The parents for which transformation products should be obtained. This can be (1) a suspect list (see suspect screening for more information), (2) the resulting output of `screenSuspects` or (3) a `compounds` annotation object. In the former two cases, the suspect (hits) are used as parents, whereas in the latter case all candidates are used as parents.
type	The type of prediction. Valid values are: `"env"`, `"ecbased"`, `"cyp450"`, `"phaseII"`, `"hgut"`, `"superbio"`, `"allHuman"`. Sets the `-b` command line option.
generations	The number of generations (steps) for the predictions. Sets the `-s` command line option. More generations may be reported, see the `Hierarchy expansion` section below.
maxExpGenerations	The maximum number of generations during hierarchy expansion, see below.
extraOpts	A `character` with extra command line options passed to the `biotransformer.jar` tool.
skipInvalid	If set to `TRUE` then the parents will be skipped (with a warning) for which insufficient information (*e.g.* SMILES) is available.
prefCalcChemProps	If `TRUE` then calculated chemical properties such as the formula and INCHIKEY are preferred over what is already present in the parent suspect list. For efficiency reasons it is recommended to set this to `TRUE`. See the `Validating and calculating chemical pr` section for more details.
neutralChemProps	If `TRUE` then the neutral form of the molecule is considered to calculate SMILES, formulae etc. Enabling this may improve feature matching when considering common adducts (*e.g.* `[M+H]+`, `[M-H]-`). See the `Validating and calculating chemical properties` section for more details.
neutralizeTPs	If `TRUE` then all resulting TP structure information is neutralized. This argument has a similar meaning as `neutralChemProps`. This is defaulted to `TRUE` for prediction algorithms, as these may output charged molecules. **NOTE:** if neutrlization results in duplicate TPs, *i.e.* when the neutral form of the TP was also generated by the algorithm, then the neutralized TP *will be removed*.
calcSims	If set to `TRUE` then structural similarities between the parent and its TPs are calculated. A minimum similarity can be obtained by using the filter method. May be useful under the assumption that parents and TPs who have a high structural similarity, also likely have a high MS/MS spectral similarity (which can be evaluated after componentization with `generateComponentsTPs`).
fpType	The type of structural fingerprint that should be calculated. See the `type` argument of the `get.fingerprint` function of **rcdk**.
fpSimMethod	The method for calculating similarities (i.e. not dissimilarity!). See the `method` argument of the `fp.sim.matrix` function of the **fingerprint** package.

MP — If TRUE then multiprocessing is enabled. Since BioTransformer supports native parallelization, additional multiprocessing generally doesn't lead to significant reduction in computational times. Furthermore, enabling multiprocessing can lead to very high CPU/RAM usage.

Details

This function uses BioTransformer to obtain transformation products. This function is called when calling generateTPs with algorithm="biotransformer".

In order to use this function the '.jar' command line utility should be installed and specified in the patRoon.path.BioTransformer option. The '.jar' file can be obtained via https://bitbucket.org/djoumbou/biotransformer/src/master. Alternatively, the **patRoonExt** package can be installed to automatically install/configure the necessary files.

An important advantage of this algorithm is that it provides structural information for generated TPs. However, this also means that if the input is from a parent suspect list or screening then either SMILES or INCHI information must be available for the parents.

Value

The TPs are stored in an object derived from the transformationProductsStructure class.

Hierarchy expansion

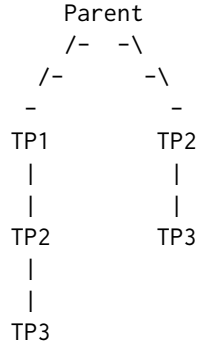
BioTransformer only reports the direct parent for a TP, not the complete pathway. For instance, consider the following results:

- parent –> TP1
- parent –> TP2
- TP1 –> TP2
- TP2 –> TP3

In this case, TP3 may be formed either as:

- parent –> TP1 –> TP2 –> TP3
- parent –> TP2 –> TP3

For this reason, **patRoon** simply expands the hierarchy and assumes that all routes are possible. For instance,



Note that this may result in pathways with more generations than defined by the `generations` argument. Thus, the `maxExpGenerations` argument is used to avoid excessive expansions.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formula in the parent suspect list are automatically validated and calculated if missing/invalid.

The internal validation/calculation process performs the following steps:

- Validation of SMILES, INCHI, INCHIKEY and formula data (if present). Invalid entries will be set to `NA`.
- If `neutralChemProps=TRUE` then chemical data (SMILES, formulae etc.) is neutralized by (de-)protonation (using the `--neutralized` option of `OpenBabel`). An additional column `molNeutralized` is added to mark those molecules that were neutralized. Note that neutralization requires either SMILES or INCHI data to be available.
- The SMILES and INCHI data are used to calculate missing or invalid SMILES, INCHI, INCHIKEY and formula data. If `prefCalcChemProps=TRUE` then existing INCHIKEY and formula data is overwritten by calculated values whenever possible.
- The chemical formulae which were *not* calculated are verified and normalized. This process may be time consuming, and is potentially largely avoided by setting `prefCalcChemProps=TRUE`.
- Neutral masses are calculated for missing values (`prefCalcChemProps=FALSE`) or whenever possible (`prefCalcChemProps=TRUE`).

Note that calculation of formulae for molecules that are isotopically labelled is currently only supported for deuterium (2H) elements.

This functionality relies heavily on OpenBabel, please make sure it is installed.

Parallelization

`generateTPsBioTransformer` uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Note

When the `parents` argument is a `compounds` object, the candidate library `identifier` is used in case the candidate has no defined `compoundName`.

References

Guha R (2007). "Chemical Informatics Functionality in R." *Journal of Statistical Software*, **18**(6).

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

Djoumbou-Feunang Y, Fiamoncini J, Gil-de-la-Fuente A, Greiner R, Manach C, Wishart DS (2019). "BioTransformer: a comprehensive computational tool for small molecule metabolism prediction and metabolite identification." *Journal of Cheminformatics*, **11**(1). doi:10.1186/s1332101803245.

Wicker J, Lorsbach T, Gutlein M, Schmid E, Latino D, Kramer S, Fenner K (2015). "enviPath - The environmental contaminant biotransformation pathway resource." *Nucleic Acids Research*, **44**(D1), D502–D508. doi:10.1093/nar/gkv1229.

See Also

`generateTPs` for more details and other algorithms.

`generateTPsCTS` *Obtain transformation products (TPs) with Chemical Transformation Simulator (CTS)*

Description

Uses Chemical Transformation Simulator (CTS) to predict TPs.

Usage

```
generateTPsCTS(
  parents,
  transLibrary,
  generations = 1,
  errorRetries = 3,
  skipInvalid = TRUE,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE,
  neutralizeTPs = TRUE,
  calcLogP = "rcdk",
  calcSims = FALSE,
  fpType = "extended",
  fpSimMethod = "tanimoto",
  parallel = TRUE
)
```

Arguments

`parents`	The parents for which transformation products should be obtained. This can be (1) a suspect list (see suspect screening for more information), (2) the resulting output of `screenSuspects` or (3) a `compounds` annotation object. In the former two cases, the suspect (hits) are used as parents, whereas in the latter case all candidates are used as parents.
`transLibrary`	A `character` specifying which transformation library should be used. Currently supported are: `"hydrolysis"`, `"abiotic_reduction"`, `"photolysis_unranked"`, `"photolysis_ranked"`, `"mammalian_metabolism"`, `"combined_abioticreduction_hydrolysis"`, `"combined_photolysis_abiotic_hydrolysis"`, `"pfas_environmental"`, `"pfas_metabolism"`.
`generations`	An `integer` that specifies the number of transformation generations to predict.

errorRetries — The maximum number of connection retries. Sets the times argument to the http::RETRY function.

skipInvalid — If set to TRUE then the parents will be skipped (with a warning) for which insufficient information (*e.g.* SMILES) is available.

prefCalcChemProps — If TRUE then calculated chemical properties such as the formula and INCHIKEY are preferred over what is already present in the parent suspect list. For efficiency reasons it is recommended to set this to TRUE. See the Validating and calculating chemical pr section for more details.

neutralChemProps — If TRUE then the neutral form of the molecule is considered to calculate SMILES, formulae etc. Enabling this may improve feature matching when considering common adducts (*e.g.* [M+H]+, [M-H]-). See the Validating and calculating chemical properties section for more details.

neutralizeTPs — If TRUE then all resulting TP structure information is neutralized. This argument has a similar meaning as neutralChemProps. This is defaulted to TRUE for prediction algorithms, as these may output charged molecules. **NOTE:** if neutrlization results in duplicate TPs, *i.e.* when the neutral form of the TP was also generated by the algorithm, then the neutralized TP *will be removed*.

calcLogP — A character specifying whether Log P values should be calculated with rcdk::get.xlogp (calcLogP="rcdk"), OpenBabel (calcLogP="obabel") or not at all (calcLogP="none"). The log P values will be calculated of parent and TPs to predict their retention order (retDir).

calcSims — If set to TRUE then structural similarities between the parent and its TPs are calculated. A minimum similarity can be obtained by using the filter method. May be useful under the assumption that parents and TPs who have a high structural similarity, also likely have a high MS/MS spectral similarity (which can be evaluated after componentization with generateComponentsTPs).

fpType — The type of structural fingerprint that should be calculated. See the type argument of the get.fingerprint function of **rcdk**.

fpSimMethod — The method for calculating similarities (i.e. not dissimilarity!). See the method argument of the fp.sim.matrix function of the **fingerprint** package.

parallel — If set to TRUE then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.

Details

This function uses CTS to obtain transformation products. This function is called when calling generateTPs with algorithm="cts".

This function uses the **httr** package to access the Web API of CTS for automatic TP prediction. Hence, an Internet connection is mandatory. Please take care to not 'abuse' the CTS servers, *e.g.* by running very large batch calculations in parallel, as this may result in rejected connections.

An important advantage of this algorithm is that it provides structural information for generated TPs. However, this also means that if the input is from a parent suspect list or screening then either SMILES or INCHI information must be available for the parents.

Value

The TPs are stored in an object derived from the `transformationProductsStructure` class.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formula in the parent suspect list are automatically validated and calculated if missing/invalid.

The internal validation/calculation process performs the following steps:

- Validation of SMILES, INCHI, INCHIKEY and formula data (if present). Invalid entries will be set to `NA`.
- If `neutralChemProps=TRUE` then chemical data (SMILES, formulae etc.) is neutralized by (de-)protonation (using the `--neutralized` option of `OpenBabel`). An additional column `molNeutralized` is added to mark those molecules that were neutralized. Note that neutralization requires either SMILES or INCHI data to be available.
- The SMILES and INCHI data are used to calculate missing or invalid SMILES, INCHI, INCHIKEY and formula data. If `prefCalcChemProps=TRUE` then existing INCHIKEY and formula data is overwritten by calculated values whenever possible.
- The chemical formulae which were *not* calculated are verified and normalized. This process may be time consuming, and is potentially largely avoided by setting `prefCalcChemProps=TRUE`.
- Neutral masses are calculated for missing values (`prefCalcChemProps=FALSE`) or whenever possible (`prefCalcChemProps=TRUE`).

Note that calculation of formulae for molecules that are isotopically labelled is currently only supported for deuterium (2H) elements.

This functionality relies heavily on OpenBabel, please make sure it is installed.

Note

When the `parents` argument is a `compounds` object, the candidate library `identifier` is used in case the candidate has no defined `compoundName`.

References

Guha R (2007). "Chemical Informatics Functionality in R." *Journal of Statistical Software*, **18**(6).

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

Wolfe K, Pope N, Parmar R, Galvin M, Stevens C, Weber E, Flaishans J, Purucker T (2016). "Chemical transformation system: Cloud based cheminformatic services to support integrated environmental modeling." *Proceedings of the 8th International Congress on Environmental Modelling and Software*.

Tebes-Stevens C, Patel JM, Jones WJ, Weber EJ (2017). "Prediction of Hydrolysis Products of Organic Chemicals under Environmental pH Conditions." *Environmental Science & Technology*, **51**(9), 5008–5016. doi:10.1021/acs.est.6b05412.

Yuan C, Tebes-Stevens C, Weber EJ (2020). "Reaction Library to Predict Direct Photochemical

Transformation Products of Environmental Organic Contaminants in Sunlit Aquatic Systems." *Environmental Science & Technology*, **54**(12), 7271–7279. doi:10.1021/acs.est.0c00484.

Yuan C, Tebes-Stevens C, Weber EJ (2021). "Prioritizing Direct Photolysis Products Predicted by the Chemical Transformation Simulator: Relative Reasoning and Absolute Ranking." *Environmental Science & Technology*, **55**(9), 5950-5958. doi:10.1021/acs.est.0c08745, PMID: 33881833, https://doi.org/10.1021/acs.est.0c08745.

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

See Also

`generateTPs` for more details and other algorithms.

The website: `https://qed.epa.gov/cts/` and the CTS User guide.

`generateTPsLibrary` *Obtain transformation products (TPs) from a library*

Description

Automatically obtains transformation products from a library.

Usage

```
generateTPsLibrary(
  parents = NULL,
  TPLibrary = NULL,
  generations = 1,
  skipInvalid = TRUE,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE,
  neutralizeTPs = FALSE,
  matchParentsBy = "InChIKey",
  matchGenerationsBy = "InChIKey",
  calcSims = FALSE,
  fpType = "extended",
  fpSimMethod = "tanimoto"
)
```

Arguments

`parents`	The parents for which transformation products should be obtained. This can be (1) a suspect list (see suspect screening for more information), (2) the resulting output of `screenSuspects` or (3) a `compounds` annotation object. In the former two cases, the suspect (hits) are used as parents, whereas in the latter case all candidates are used as parents. If `NULL` then TPs for all parents in the library are obtained.

`TPLibrary` If `NULL`, a default PubChem based library is used. Otherwise, `TPLibrary` should be a `data.frame`. See the details below.

`generations` An `integer` that specifies the number of transformation generations. TPs for subsequent iterations obtained by repeating the library search where the TPs from the previous generation are considered parents.

`skipInvalid` If set to `TRUE` then the parents will be skipped (with a warning) for which insufficient information (*e.g.* SMILES) is available.

`prefCalcChemProps`
If `TRUE` then calculated chemical properties such as the formula and INCHIKEY are preferred over what is already present in the parent suspect list. For efficiency reasons it is recommended to set this to `TRUE`. See the `Validating and calculating chemical pr` section for more details.

`neutralChemProps`
If `TRUE` then the neutral form of the molecule is considered to calculate SMILES, formulae etc. Enabling this may improve feature matching when considering common adducts (*e.g.* `[M+H]+`, `[M-H]-`). See the `Validating and calculating chemical properties` section for more details.

`neutralizeTPs` If `TRUE` then all resulting TP structure information is neutralized. This argument has a similar meaning as `neutralChemProps`. This is defaulted to `TRUE` for prediction algorithms, as these may output charged molecules. **NOTE:** if neutrlization results in duplicate TPs, *i.e.* when the neutral form of the TP was also generated by the algorithm, then the neutralized TP *will be removed*.

`matchParentsBy` A `character` that specifies how the input parents are matched with the data from the TP library. Valid options are: `"InChIKey"`, `"InChIKey1"`, `"InChI"`, `"SMILES"`, `"formula"`, `"name"`. If the parent from the TP library is matched with multiple input parents then only the first is considered.

`matchGenerationsBy`
Similar to `matchParentsBy`, but specifies how parents/TPs are matched when `generations>1`.

`calcSims` If set to `TRUE` then structural similarities between the parent and its TPs are calculated. A minimum similarity can be obtained by using the filter method. May be useful under the assumption that parents and TPs who have a high structural similarity, also likely have a high MS/MS spectral similarity (which can be evaluated after componentization with `generateComponentsTPs`).

`fpType` The type of structural fingerprint that should be calculated. See the `type` argument of the `get.fingerprint` function of **rcdk**.

`fpSimMethod` The method for calculating similarities (i.e. not dissimilarity!). See the `method` argument of the `fp.sim.matrix` function of the **fingerprint** package.

Details

This function uses a library to obtain transformation products. This function is called when calling `generateTPs` with `algorithm="library"`.

By default, a library is used that is based on data from PubChem. However, it also possible to use your own library.

An important advantage of this algorithm is that it provides structural information for generated TPs. However, this also means that if the input is from a parent suspect list or screening then either SMILES or INCHI information must be available for the parents.

Value

The TPs are stored in an object derived from the `transformationProductsStructure` class.

TP libraries

The `TPLibrary` argument is used to specify a custom TP library. This should be a `data.frame` where each row specifies a TP for a parent, with the following columns:

- `parent_name` and `TP_name`: The name of the parent/TP.
- `parent_SMILES` and `TP_SMILES` The SMILES of the parent/TP structure.
- `retDir` The retention direction of the TP compared to its parent: '`-1`' (elutes before), '`1`' (elutes after) or '`0`' (elutes similarly or unknown). If not specified then the `log P` values below may be used to calculate retention time directions. (**optional**)
- `parent_LogP` and `TP_LogP` The `log P` values for the parent/TP. (**optional**)
- `LogPDiff` The difference between parent and TP `Log P` values. Ignored if *both* `parent_LogP` and `TP_LogP` are specified. (**optional**)

Other columns are allowed, and will be included in the final object. Multiple TPs for a single parent are specified by repeating the value within `parent_` columns.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formula in the parent suspect list are automatically validated and calculated if missing/invalid.

The internal validation/calculation process performs the following steps:

- Validation of SMILES, INCHI, INCHIKEY and formula data (if present). Invalid entries will be set to `NA`.
- If `neutralChemProps=TRUE` then chemical data (SMILES, formulae etc.) is neutralized by (de-)protonation (using the `--neutralized` option of `OpenBabel`). An additional column `molNeutralized` is added to mark those molecules that were neutralized. Note that neutralization requires either SMILES or INCHI data to be available.
- The SMILES and INCHI data are used to calculate missing or invalid SMILES, INCHI, INCHIKEY and formula data. If `prefCalcChemProps=TRUE` then existing INCHIKEY and formula data is overwritten by calculated values whenever possible.
- The chemical formulae which were *not* calculated are verified and normalized. This process may be time consuming, and is potentially largely avoided by setting `prefCalcChemProps=TRUE`.
- Neutral masses are calculated for missing values (`prefCalcChemProps=FALSE`) or whenever possible (`prefCalcChemProps=TRUE`).

Note that calculation of formulae for molecules that are isotopically labelled is currently only supported for deuterium (2H) elements.

This functionality relies heavily on OpenBabel, please make sure it is installed.

Note

When the parents argument is a compounds object, the candidate library identifier is used in case the candidate has no defined compoundName.

References

Guha R (2007). "Chemical Informatics Functionality in R." *Journal of Statistical Software*, **18**(6).

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

See Also

generateTPs for more details and other algorithms.

---

generateTPsLibraryFormula *Obtain transformation products (TPs) from a library with formula data*

Description

Automatically obtains transformation products from a library with formula data.

Usage

```
generateTPsLibraryFormula(
  parents = NULL,
  TPLibrary,
  generations = 1,
  skipInvalid = TRUE,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE,
  matchParentsBy = "name",
  matchGenerationsBy = "name"
)
```

Arguments

parents	The parents for which transformation products should be obtained. This should be either a suspect list (see suspect screening for more information) or the resulting output of screenSuspects. The suspect (hits) are used as parents. If NULL then TPs for all parents in the library are obtained.
TPLibrary	A data.frame. See the details below.
generations	An integer that specifies the number of transformation generations. TPs for subsequent iterations obtained by repeating the library search where the TPs from the previous generation are considered parents.

skipInvalid If set to TRUE then the parents will be skipped (with a warning) for which insufficient information (*e.g.* SMILES) is available.

prefCalcChemProps
If TRUE then calculated chemical properties such as the formula and INCHIKEY are preferred over what is already present in the parent suspect list. For efficiency reasons it is recommended to set this to TRUE. See the Validating and calculating chemical pr section for more details.

neutralChemProps
If TRUE then the neutral form of the molecule is considered to calculate SMILES, formulae etc. Enabling this may improve feature matching when considering common adducts (*e.g.* [M+H]+, [M-H]-). See the Validating and calculating chemical properties section for more details.

matchParentsBy A character that specifies how the input parents are matched with the data from the TP library. Valid options are: "InChIKey", "InChIKey1", "InChI", "SMILES", "formula", "name". If the parent from the TP library is matched with multiple input parents then only the first is considered.

matchGenerationsBy
Similar to matchParentsBy, but specifies how parents/TPs are matched when generations>1.

Details

This function uses a library to obtain transformation products. This function is called when calling generateTPs with algorithm="library_formula".

This function is similar to generateTPsLibrary, however, it only require formula information of the parent and TPs.

Value

The TPs are stored in an object derived from the transformationProductsFormula class.

TP libraries

The TPLibrary argument is used to specify a custom TP library. This should be a data.frame where each row specifies a TP for a parent, with the following columns:

- parent_name and TP_name: The name of the parent/TP.
- parent_formula and TP_formula The formula of the parent/TP structure.
- retDir The retention direction of the TP compared to its parent: '-1' (elutes before), '1' (elutes after) or '0' (elutes similarly or unknown). If not specified then the log P values below may be used to calculate retention time directions. (**optional**)
- parent_LogP and TP_LogP The log P values for the parent/TP. (**optional**)
- LogPDiff The difference between parent and TP Log P values. Ignored if *both* parent_LogP and TP_LogP are specified. (**optional**)

Other columns are allowed, and will be included in the final object. Multiple TPs for a single parent are specified by repeating the value within parent_ columns.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formula in the parent suspect list are automatically validated and calculated if missing/invalid.

The internal validation/calculation process performs the following steps:

- Validation of SMILES, INCHI, INCHIKEY and formula data (if present). Invalid entries will be set to NA.
- If `neutralChemProps=TRUE` then chemical data (SMILES, formulae etc.) is neutralized by (de-)protonation (using the `--neutralized` option of `OpenBabel`). An additional column `molNeutralized` is added to mark those molecules that were neutralized. Note that neutralization requires either SMILES or INCHI data to be available.
- The SMILES and INCHI data are used to calculate missing or invalid SMILES, INCHI, INCHIKEY and formula data. If `prefCalcChemProps=TRUE` then existing INCHIKEY and formula data is overwritten by calculated values whenever possible.
- The chemical formulae which were *not* calculated are verified and normalized. This process may be time consuming, and is potentially largely avoided by setting `prefCalcChemProps=TRUE`.
- Neutral masses are calculated for missing values (`prefCalcChemProps=FALSE`) or whenever possible (`prefCalcChemProps=TRUE`).

Note that calculation of formulae for molecules that are isotopically labelled is currently only supported for deuterium (2H) elements.

This functionality relies heavily on OpenBabel, please make sure it is installed.

Note

Unlike `generateTPsLibrary`, this function defaults the `matchParentsBy` and `matchGenerationsBy` arguments to `"name"`. While matching by `formula` is also possible, it is likely that duplicate parent formulae (*i.e.* isomers) are present in `parents` and/or `TPLibrary`, making matching by formula unsuitable. However, if you are sure that no duplicate formulae are present, it may be better to set the matching method to `"formula"`.

References

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

See Also

`generateTPs` for more details and other algorithms.

`generateTPsLibrary` to generate TPs from a library that contains structural information.

`genFormulaTPLibrary` to automatically generate formula TP libraries.

`generateTPsLogic` *Obtain transformation products (TPs) with metabolic logic*

Description

Automatically calculate potential transformation products with *metabolic logic*.

Usage

```
generateTPsLogic(fGroups, minMass = 40, ...)

## S4 method for signature 'featureGroups'
generateTPsLogic(fGroups, minMass = 40, adduct = NULL, transformations = NULL)

## S4 method for signature 'featureGroupsSet'
generateTPsLogic(fGroups, minMass = 40, transformations = NULL)
```

Arguments

`fGroups`	A `featureGroups` object for which TPs should be calculated.
`minMass`	A `numeric` that specifies the minimum mass of calculated TPs. If below this mass it will be removed.
`...`	Further arguments specified to the methods.
`adduct`	An `adduct` object (or something that can be converted to it with `as.adduct`). Examples: `"[M-H]-"`, `"[M+Na]+"`. If the `featureGroups` object has adduct annotations then these are used if `adducts=NULL`. (**sets workflow**) The `adduct` argument is not supported for sets workflows, since the adduct annotations will then always be used.
`transformations`	A `data.frame` with transformation reactions to be used for calculating the TPs (see details below). If `NULL`, a default table from Schollee *et al.* is used (see references).

Details

This function uses metabolic logic to obtain transformation products. This function is called when calling `generateTPs` with `algorithm="logic"`.

With this algorithm TPs are predicted from common (environmental) chemical reactions, such as hydroxylation, demethylation etc. The generated TPs result from calculating the mass differences between a parent feature after it underwent the reaction. While this only results in little information on chemical properties of the TP, an advantage of this method is that it does not rely on structural information of the parent, which may be unknown in a full non-target analysis.

Value

A `transformationProducts` (derived) object containing all generated TPs.

Transformation reactions

The `transformations` argument specifies custom rules to calculate transformation products. This should be a `data.frame` with the following columns:

- `transformation` The name of the chemical transformation
- `add` The elements that are added by this reaction (*e.g.* `"O"`).
- `sub` The elements that are removed by this reaction (*e.g.* `"H2O"`).
- `retDir` The expected retention time direction relative to the parent (assuming a reversed phase like LC separation). Valid values are: '`-1`' (elutes before the parent), '`1`' (elutes after the parent) or '`0`' (no significant change or unknown).

Source

The algorithms using transformation reactions are directly based on the work done by Schollee *et al.* (see references).

References

Schollee JE, Schymanski EL, Avak SE, Loos M, Hollender J (2015). "Prioritizing Unknown Transformation Products from Biologically-Treated Wastewater Using High-Resolution Mass Spectrometry, Multivariate Statistics, and Metabolic Logic." *Analytical Chemistry*, **87**(24), 12121–12129. doi:10.1021/acs.analchem.5b02905.

See Also

`generateTPs` for more details and other algorithms.

`generics` *Miscellaneous generics*

Description

Various (S4) generic functions providing a common interface for common tasks such as plotting and filtering data. The actual functionality and function arguments are often specific for the implemented methods, for this reason, please refer to the linked method documentation for each generic.

Usage

```
adducts(obj, ...)

adducts(obj, ...) <- value

algorithm(obj)

analysisInfo(obj)
```

```
analyses(obj)

annotatedPeakList(obj, ...)

annotations(obj, ...)

calculatePeakQualities(obj, weights = NULL, flatnessFactor = 0.05, ...)

clusterProperties(obj)

clusters(obj)

consensus(obj, ...)

convertToMFDB(TPs, out, ...)

convertToSuspects(obj, ...)

cutClusters(obj)

defaultExclNormScores(obj)

export(obj, type, out, ...)

featureTable(obj, ...)

filter(obj, ...)

getBPCs(obj, ...)

getFeatures(obj)

getMCS(obj, ...)

getTICs(obj, ...)

groupNames(obj)

plotBPCs(obj, ...)

plotChord(obj, addSelfLinks = FALSE, addRetMzPlots = TRUE, ...)

plotChroms(obj, ...)

plotGraph(obj, ...)

plotInt(obj, ...)
```

```
plotScores(obj, ...)

plotSilhouettes(obj, kSeq, ...)

plotSpectrum(obj, ...)

plotStructure(obj, ...)

plotTICs(obj, ...)

plotVenn(obj, ...)

plotUpSet(obj, ...)

predictRespFactors(obj, ...)

predictTox(obj, ...)

delete(obj, ...)

plotVolcano(obj, ...)

replicateGroups(obj)

setObjects(obj)

sets(obj)

treeCut(obj, k = NULL, h = NULL, ...)

treeCutDynamic(
  obj,
  maxTreeHeight = 1,
  deepSplit = TRUE,
  minModuleSize = 1,
  ...
)

unset(obj, set)
```

Arguments

obj	The object the generic should be applied to.
...	Any further method specific arguments. See method documentation for details.
value	The replacement value.
weights, flatnessFactor	See method documentation.
TPs	The transformationProducts derived object.

out	Output file.
type	The export type.
addSelfLinks	If TRUE then 'self-links' are added which represent non-shared data.
addRetMzPlots	Set to TRUE to enable *m/z vs* retention time scatter plots.
kSeq	An integer vector containing the sequence that should be used for average silhouette width calculation.
k, h	Desired numbers of clusters. See cutree.
maxTreeHeight, deepSplit, minModuleSize	Arguments used by cutreeDynamicTree.
set	The name of the set.

Details

adducts returns assigned adducts of the object.

- Methods are defined for: featureGroups; featureGroupsSet.

adducts<- sets adducts of the object.

- Methods are defined for: featureGroups; featureGroupsSet.

algorithm returns the algorithm that was used to generate the object.

- Methods are defined for: optimizationResult; workflowStep.

analysisInfo returns the analysis information from an object.

- Methods are defined for: featureGroups; features; MSPeakListsSet.

analyses returns a character vector with the analyses for which data is present in this object.

- Methods are defined for: featureGroups; features; formulas; MSPeakLists.

annotatedPeakList returns an annotated MS peak list.

- Methods are defined for: compounds; compoundsSet; formulas; formulasSet.

annotations returns annotations.

- Methods are defined for: featureAnnotations; featureGroups; formulas.

calculatePeakQualities calculates chromatographic peak qualities and scores.

- Methods are defined for: featureGroups; features.

clusterProperties Obtain a list with properties of the generated cluster(s).

- Methods are defined for: componentsClust; compoundsCluster.

clusters Obtain clustering object(s).

- Methods are defined for: componentsClust; compoundsCluster.

consensus combines and merges data from various algorithms to generate a consensus.

- Methods are defined for: components; componentsSet; compounds; compoundsSet; featureGroupsComparison; featureGroupsComparisonSet; formulas; formulasSet; transformationProductsStructure.

convertToMFDB Exports the object to a local database that can be used with MetFrag.

- Methods are defined for: .

convertToSuspects Converts an object to a suspect list.

- Methods are defined for: MSLibrary; transformationProducts.

cutClusters Returns assigned cluster indices of a cut cluster.

- Methods are defined for: componentsClust; compoundsCluster.

defaultExclNormScores Returns default scorings that are excluded from normalization.

- Methods are defined for: compounds; formulas.

export exports workflow data to a given format.

- Methods are defined for: featureGroups; featureGroupsSet; MSLibrary.

featureTable returns feature information.

- Methods are defined for: featureGroups; featureGroupsSet; features.

filter provides various functionality to do post-filtering of data.

- Methods are defined for: components; componentsSet; componentsTPs; compounds; compoundsSet; featureAnnotations; featureGroups; featureGroupsScreening; featureGroupsScreeningSet; featureGroupsSet; features; featuresSet; formulasSet; MSLibrary; MSPeakLists; MSPeakListsSet; transformationProducts; transformationProductsStructure.

getBPCs gets base peak chromatogram(s).

- Methods are defined for: featureGroups; features.

getFeatures returns the object's features object.

- Methods are defined for: featureGroups.

getMCS Calculates the maximum common substructure.

- Methods are defined for: compounds; compoundsCluster.

getTICs gets total ion chromatogram(s).

- Methods are defined for: featureGroups; features.

groupNames returns a character vector with the names of the feature groups for which data is present in this object.

- Methods are defined for: components; compoundsCluster; featureAnnotations; featureGroups; MSPeakLists.

plotBPCs plots base peak chromatogram(s).

- Methods are defined for: featureGroups; features.

plotChord plots a Chord diagram to assess overlapping data.

- Methods are defined for: featureGroups; featureGroupsComparison.

plotChroms plots extracted ion chromatogram(s).

- Methods are defined for: components; featureGroups.

plotGraph Plots an interactive network graph.

- Methods are defined for: componentsNT; componentsNTSet; componentsTPs; featureGroups; featureGroupsSet; transformationProductsFormula; transformationProductsStructure.

plotInt plots the intensity of all contained features.

- Methods are defined for: componentsIntClust; featureGroups; featureGroupsSet.

plotScores plots candidate scorings.

- Methods are defined for: compounds; formulas.

plotSilhouettes plots silhouette widths to evaluate the desired cluster size.

- Methods are defined for: componentsClust; compoundsCluster.

plotSpectrum plots a (annotated) spectrum.

- Methods are defined for: components; compounds; compoundsSet; formulas; formulasSet; MSPeakLists; MSPeakListsSet.

plotStructure plots a chemical structure.

- Methods are defined for: compounds; compoundsCluster.

plotTICs plots total ion chromatogram(s).

- Methods are defined for: featureGroups; features.

plotVenn plots a Venn diagram to assess unique and overlapping data.

- Methods are defined for: featureAnnotations; featureGroups; featureGroupsComparison; featureGroupsSet; transformationProductsStructure.

`plotUpSet` plots an UpSet diagram to assess unique and overlapping data.

- Methods are defined for: `featureAnnotations`; `featureGroups`; `featureGroupsComparison`; `transformationProductsStructure`.

`predictRespFactors` Prediction of response factors.

- Methods are defined for: `compounds`; `compoundsSet`; `compoundsSIRIUS`; `featureGroupsScreening`; `featureGroupsScreeningSet`; `formulasSet`; `formulasSIRIUS`.

`predictTox` Prediction of toxicity values.

- Methods are defined for: `compounds`; `compoundsSet`; `compoundsSIRIUS`; `featureGroupsScreening`; `featureGroupsScreeningSet`; `formulasSet`; `formulasSIRIUS`.

`delete` Deletes results.

- Methods are defined for: `components`; `componentsClust`; `componentsSet`; `compoundsSet`; `compoundsSIRIUS`; `featureAnnotations`; `featureGroups`; `featureGroupsKPIC2`; `featureGroupsScreening`; `featureGroupsScreeningSet`; `featureGroupsSet`; `featureGroupsXCMS`; `featureGroupsXCMS3`; `features`; `featuresKPIC2`; `featuresXCMS`; `featuresXCMS3`; `formulas`; `formulasSet`; `formulasSIRIUS`; `MSLibrary`; `MSPeakLists`; `MSPeakListsSet`; `transformationProducts`.

`plotVolcano` plots a volcano plot.

- Methods are defined for: `featureGroups`.

`replicateGroups` returns a `character` vector with the analyses for which data is present in this object.

- Methods are defined for: `featureGroups`; `features`.

`setObjects` returns the *set objects* of this object. See the documentation of `workflowStepSet`.

- Methods are defined for: `workflowStepSet`.

`sets` returns the names of the sets inside this object. See the documentation for sets workflows.

- Methods are defined for: `featureGroupsSet`; `featuresSet`; `workflowStepSet`.

`treeCut` Manually cut a cluster.

- Methods are defined for: `componentsClust`; `compoundsCluster`.

`treeCutDynamic` Automatically cut a cluster.

- Methods are defined for: `componentsClust`; `compoundsCluster`.

`unset` Converts this object to a regular non-set object. See the documentation for sets workflows.

- Methods are defined for: `componentsNTSet`; `componentsSet`; `compoundsConsensusSet`; `compoundsSet`; `featureGroupsScreeningSet`; `featureGroupsSet`; `featuresSet`; `formulasConsensusSet`; `formulasSet`; `MSPeakListsSet`.

Other generics

Below are methods that are defined for existing generics (*e.g.* defined in base). Please see method specific documentation for more details.

[Subsets data within an object.

- Methods are defined for: components,ANY,ANY,missing; componentsSet,ANY,ANY,missing; compoundsCluster,ANY,missing,missing; compoundsSet,ANY,missing,missing; featureAnnotations,ANY,mi featureGroups,ANY,ANY,missing; featureGroupsComparison,ANY,missing,missing; featureGroupsScreenin featureGroupsScreeningSet,ANY,ANY,missing; featureGroupsSet,ANY,ANY,missing; features,ANY,missing,missing; featuresSet,ANY,missing,missing; formulasSet,ANY,missing,missing; MSLibrary,ANY,missing,missing; MSPeakLists,ANY,ANY,missing; MSPeakListsSet,ANY,ANY,missing; transformationProducts,ANY,missing,missing.

[[Extract data from an object.

- Methods are defined for: components,ANY,ANY; featureAnnotations,ANY,missing; featureGroups,ANY,ANY; featureGroupsComparison,ANY,missing; features,ANY,missing; formulas,ANY,ANY; MSLibrary,ANY,missing; MSPeakLists,ANY,ANY; transformationProducts,ANY,missing.

$ Extract data from an object.

- Methods are defined for: components; featureAnnotations; featureGroups; featureGroupsComparison; features; MSLibrary; MSPeakLists; transformationProducts.

as.data.table Converts an object to a table (data.table).

- Methods are defined for: components; componentsTPs; featureAnnotations; featureGroups; featureGroupsScreening; featureGroupsScreeningSet; features; featuresSet; formulas; MSLibrary; MSPeakLists; MSPeakListsSet; transformationProducts; workflowStep.

as.data.frame Converts an object to a table (data.frame).

- Methods are defined for: workflowStep.

length Returns the length of an object.

- Methods are defined for: components; compoundsCluster; featureAnnotations; featureGroups; featureGroupsComparison; features; MSLibrary; MSPeakLists; optimizationResult; transformationProducts.

lengths Returns the lengths of elements within this object.

- Methods are defined for: compoundsCluster; optimizationResult.

names Return names for this object.

- Methods are defined for: components; featureGroups; featureGroupsComparison; MSLibrary; transformationProducts.

plot Generates a plot for an object.

- Methods are defined for: componentsClust,missing; compoundsCluster,missing; featureGroups,missing; featureGroupsComparison,missing; optimizationResult,missing.

show Prints information about this object.

- Methods are defined for: adduct; components; componentsFeatures; componentsSet; compounds; compoundsCluster; compoundsSet; featureGroups; featureGroupsScreening; featureGroupsScreeningSet; featureGroupsSet; features; featuresSet; formulas; formulasSet; MSLibrary; MSPeakLists; MSPeakListsSet; optimizationResult; transformationProducts; workflowStep; workflowStepSet.

---

genFormulaTPLibrary *Automatically generate a transformation product library with formula data.*

Description

Functionality to automatically generate a TP library with formula data from a set of transformation rules, which can be used with generateTPsLibraryFormula. TP calculation will be skipped if the transformation involves subtraction of elements not present in the parent.

Usage

```
genFormulaTPLibrary(
  parents,
  transformations = NULL,
  minMass = 40,
  generations = 1,
  skipInvalid = TRUE,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE
)
```

Arguments

parents	The parents to which the given transformation rules should be used to generate the TP library. Should be either a suspect list (see suspect screening for more information) or the resulting output of screenSuspects.
transformations	A data.frame with transformation reactions to be used for calculating the TPs (see details below). If NULL, a default table from Schollee *et al.* is used (see references).
minMass	The minimum mass for a TP to be kept.
generations	An integer that specifies the number of transformation generations that should be calculated. If generations>1 then TPs are calculated by applying the transformation rules to the TPs generated in the previous generation.

skipInvalid Set to TRUE to skip parents without formula information. Otherwise an error is thrown.

prefCalcChemProps
If TRUE then calculated chemical properties such as the formula and INCHIKEY are preferred over what is already present in the parent suspect list. For efficiency reasons it is recommended to set this to TRUE. See the Validating and calculating chemical pr section for more details.

neutralChemProps
If TRUE then the neutral form of the molecule is considered to calculate SMILES, formulae etc. Enabling this may improve feature matching when considering common adducts (*e.g.* [M+H]+, [M-H]-). See the Validating and calculating chemical properties section for more details.

Value

A data.table that is suitable for the TPLibrary argument to generateTPsLibraryFormula.

Transformation reactions

The transformations argument specifies custom rules to calculate transformation products. This should be a data.frame with the following columns:

- transformation The name of the chemical transformation
- add The elements that are added by this reaction (*e.g.* "O").
- sub The elements that are removed by this reaction (*e.g.* "H2O").
- retDir The expected retention time direction relative to the parent (assuming a reversed phase like LC separation). Valid values are: '-1' (elutes before the parent), '1' (elutes after the parent) or '0' (no significant change or unknown).

Source

The algorithms using transformation reactions are directly based on the work done by Schollee *et al.* (see references).

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formula in the parent suspect list are automatically validated and calculated if missing/invalid.

The internal validation/calculation process performs the following steps:

- Validation of SMILES, INCHI, INCHIKEY and formula data (if present). Invalid entries will be set to NA.
- If neutralChemProps=TRUE then chemical data (SMILES, formulae etc.) is neutralized by (de-)protonation (using the --neutralized option of OpenBabel). An additional column molNeutralized is added to mark those molecules that were neutralized. Note that neutralization requires either SMILES or INCHI data to be available.
- The SMILES and INCHI data are used to calculate missing or invalid SMILES, INCHI, INCHIKEY and formula data. If prefCalcChemProps=TRUE then existing INCHIKEY and formula data is overwritten by calculated values whenever possible.

- The chemical formulae which were *not* calculated are verified and normalized. This process may be time consuming, and is potentially largely avoided by setting `prefCalcChemProps=TRUE`.
- Neutral masses are calculated for missing values (`prefCalcChemProps=FALSE`) or whenever possible (`prefCalcChemProps=TRUE`).

Note that calculation of formulae for molecules that are isotopically labelled is currently only supported for deuterium (2H) elements.

This functionality relies heavily on OpenBabel, please make sure it is installed.

References

Schollee JE, Schymanski EL, Avak SE, Loos M, Hollender J (2015). "Prioritizing Unknown Transformation Products from Biologically-Treated Wastewater Using High-Resolution Mass Spectrometry, Multivariate Statistics, and Metabolic Logic." *Analytical Chemistry*, **87**(24), 12121–12129. doi:10.1021/acs.analchem.5b02905.

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

See Also

`generateTPsLibraryFormula` and `generateTPsLogic`

`getDefAvgPListParams` *Parameters for averaging MS peak list data*

Description

Create parameter lists for averaging MS peak list data.

Usage

```
getDefAvgPListParams(...)
```

Arguments

...	Optional named arguments that override defaults.

Details

The parameters set used for averaging peak lists are set by the `avgFeatParams` and `avgFGroupParams` arguments to `generateMSPeakLists` and its related algorithm specific functions. The parameters are specified as a named `list` with the following values:

- `clusterMzWindow` *m/z* window (in Da) used for clustering *m/z* values when spectra are averaged. For `method="hclust"` this corresponds to the cluster height, while for `method="distance"` this value is used to find nearby masses (+/- window). Too small windows will prevent clustering *m/z* values (thus erroneously treating equal masses along spectra as different), whereas too big windows may cluster unrelated *m/z* values from different or even the same spectrum together.

- `topMost` Only retain this maximum number of MS peaks when generating averaged spectra. Lowering this number may exclude more irrelevant (noisy) MS peaks and decrease processing time, whereas higher values may avoid excluding lower intense MS peaks that may still be of interest.
- `minIntensityPre` MS peaks with intensities below this value will be removed (applied prior to selection by `topMost`) before averaging.
- `minIntensityPost` MS peaks with intensities below this value will be removed after averaging.
- `avgFun` Function that is used to calculate average *m/z* values.
- `method` Method used for producing averaged MS spectra. Valid values are `"hclust"`, used for hierarchical clustering (using the **fastcluster** package), and `"distance"`, to use the between peak distance. The latter method may reduces processing time and memory requirements, at the potential cost of reduced accuracy.
- `pruneMissingPrecursorMS` For MS data only: if `TRUE` then peak lists without a precursor peak are removed. Note that even when this is set to `FALSE`, functionality that relies on MS (not MS/MS) peak lists (*e.g.* formulae calulcation) will still skip calculation if a precursor is not found.
- `retainPrecursorMSMS` For MS/MS data only: if `TRUE` then always retain the precursor mass peak even if is not amongst the `topMost` peaks. Note that MS precursor mass peaks are always kept. Furthermore, note that precursor peaks in both MS and MS/MS data may still be removed by intensity thresholds (this is unlike the `filter` method function).

The `getDefAvgPListParams` function can be used to generate a default parameter list. The current defaults are:

`clusterMzWindow=0.005`; `topMost=50`; `minIntensityPre=500`; `minIntensityPost=500`; `avgFun=mean`; `method="hclust"`; `pruneMissingPrecursorMS=TRUE`; `retainPrecursorMSMS=TRUE`

Value

`getDefAvgPListParams` returns a `list` with the peak list averaging parameters.

Source

Averaging of mass spectra algorithms used by are based on the msProcess R package (now archived on CRAN).

Note

With Bruker algorithms these parameters only control generation of feature groups averaged peak lists: how peak lists for features are generated is controlled by DataAnalysis.

References

Müllner D (2013). "fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python." *Journal of Statistical Software*, **53**(9), 1–18. doi:10.18637/jss.v053.i09.

getEICs *Obtains extracted ion chromatograms (EICs)*

Description

This function generates one or more EIC(s) for given retention time and *m/z* ranges.

Usage

```
getEICs(file, ranges)
```

Arguments

file	The file path to the sample analysis data file ('.mzXML' or '.mzML').
ranges	A data.frame with numeric columns "retmin", "retmin", "mzmin", "mzmax" with the lower/upper ranges of the retention time and *m/z*.

Value

A list with EIC data for each of the rows in ranges.

getFCParams *Fold change calculation*

Description

Fold change calculation

Usage

```
getFCParams(rGroups, ...)
```

Arguments

rGroups	A character vector with the names of the two replicate groups to be compared.
...	Optional named arguments that override defaults.

Details

Fold change calculation can be used to easily identify significant changes between replicate groups. The calculation process is configured through a paramater list, which can be constructed with the `getFCParams` function. The parameter list has the following entries:

- `rGroups` the name of the two replicate groups to compare (taken from the `rGroups` argument to `getFCParams`).
- `thresholdFC`: the threshold log FC for a feature group to be classified as increasing/decreasing.
- `thresholdPV`: the threshold log P for a feature group to be significantly different.
- `zeroMethod`,`zeroValue`: how to handle zero values when calculating the FC: `add` adds an offset to zero values, `"fixed"` sets zero values to a fixed number and `"omit"` removes zero data. The number that is added/set by the former two options is defined by `zeroValue`.
- `PVTestFunc`: a function that is used to calculate P values (usually using `t.test`).
- `PVAdjFunc`: a function that is used to adjust P values (usually using `p.adjust`)

Author(s)

The code to calculate and plot Fold change data was created by Bas van de Velde.

See Also

`featureGroups-class` and feature-plotting

---

getPICSet *Conversion to KPIC2 objects*

Description

Converts a `features` object to an **KPIC** object.

Usage

```
getPICSet(obj, ...)

## S4 method for signature 'features'
getPICSet(obj, loadRawData = TRUE)

## S4 method for signature 'featuresKPIC2'
getPICSet(obj, ...)
```

Arguments

`obj`	The `features` object that should be converted.
`...`	Ignored
`loadRawData`	Set to `TRUE` if analyses are available as `mzXML` or `mzML` files. Otherwise MS data is not loaded, and some dummy data (*e.g.* file paths) is used in the returned object.

getXCMSSet *Conversion to XCMS objects*

Description

Converts a features or featureGroups object to an xcmsSet or XCMSnExp object.

Usage

```
getXCMSSet(obj, verbose = TRUE, ...)

getXCMSnExp(obj, verbose = TRUE, ...)

## S4 method for signature 'features'
getXCMSSet(obj, verbose, loadRawData)

## S4 method for signature 'featuresXCMS'
getXCMSSet(obj, verbose = TRUE, ...)

## S4 method for signature 'featureGroups'
getXCMSSet(obj, verbose, loadRawData)

## S4 method for signature 'featureGroupsXCMS'
getXCMSSet(obj, verbose, loadRawData)

## S4 method for signature 'featuresSet'
getXCMSSet(obj, ..., set)

## S4 method for signature 'featureGroupsSet'
getXCMSSet(obj, ..., set)

## S4 method for signature 'features'
getXCMSnExp(obj, verbose, loadRawData)

## S4 method for signature 'featuresXCMS3'
getXCMSnExp(obj, verbose = TRUE, ...)

## S4 method for signature 'featureGroups'
getXCMSnExp(obj, verbose, loadRawData)

## S4 method for signature 'featureGroupsXCMS3'
getXCMSnExp(obj, verbose, loadRawData)

## S4 method for signature 'featuresSet'
getXCMSnExp(obj, ..., set)

## S4 method for signature 'featureGroupsSet'
```

```
getXCMSnExp(obj, ..., set)
```

Arguments

obj	The object that should be converted.
verbose	If FALSE then no text output is shown.
...	(**sets workflow**) Further arguments passed to non-sets method. Otherwise ignored.
loadRawData	Set to TRUE if analyses are available as mzXML or mzML files. Otherwise MS data is not loaded, and some dummy data (*e.g.* file paths) is used in the returned object.
set	(**sets workflow**) The name of the set to be exported.

Sets workflows

In a sets workflow, unset is used to convert the feature (group) data before the object is exported.

---

groupFeatures *Grouping of features*

Description

Group equal features across analyses.

Usage

```
groupFeatures(obj, algorithm, ...)

## S4 method for signature 'features'
groupFeatures(obj, algorithm, ..., verbose = TRUE)

## S4 method for signature 'data.frame'
groupFeatures(obj, algorithm, ..., verbose = TRUE)
```

Arguments

obj	Either a features object to be grouped, or a data.frame with analysis info to be passed to groupFeaturesSIRIUS
algorithm	A character that specifies the algorithm to be used: either "openms", "xcms", "xcms3" or "kpic2" (features method), or "sirius" (data.frame method).
...	Further parameters passed to the selected grouping algorithm.
verbose	if FALSE then no text output will be shown.

Details

After features have been found, the next step is to align and group them across analyses. This process is necessary to allow comparison of features between multiple analyses, which otherwise would be difficult due to small deviations in retention and mass data. Thus, algorithms of 'feature groupers' are used to collect features with similar retention and mass data. In addition, advanced retention time alignment algorithms exist to enhance grouping of features even with relative large retention time deviations (*e.g.* possibly observed from analyses collected over a long period). Like findFeatures, various algorithms are supported which may have many parameters that can be fine-tuned. This fine-tuning is likely to be necessary, since optimal settings often depend on applied methodology and instrumentation.

`groupFeatures` is a generic function that will groupFeatures by one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as `groupFeaturesOpenMS` and `groupFeaturesXCMS3`. While these functions may be called directly, `groupFeatures` provides a generic interface and is therefore usually preferred.

The `data.frame` method for `groupFeatures` is a special case that currently only supports the `"sirius"` algorithm.

Value

An object of a class which is derived from `featureGroups`.

See Also

The `featureGroups` output class and its methods and the algorithm specific functions: `groupFeaturesOpenMS`, `groupFeaturesXCMS`, `groupFeaturesXCMS3`, `groupFeaturesKPIC2`, `groupFeaturesSIRIUS`

---

groupFeaturesKPIC2 *Group features using KPIC2*

Description

Uses the the KPIC2 R package for grouping of features.

Usage

```
groupFeaturesKPIC2(feat, ...)

## S4 method for signature 'features'
groupFeaturesKPIC2(
  feat,
  rtalign = TRUE,
  loadRawData = TRUE,
  groupArgs = list(tolerance = c(0.005, 12)),
  alignArgs = list(),
  verbose = TRUE
)
```

```
## S4 method for signature 'featuresSet'
groupFeaturesKPIC2(
  feat,
  groupArgs = list(tolerance = c(0.005, 12)),
  verbose = TRUE
)
```

Arguments

feat	The features object with the features to be grouped.
...	Further parameters passed to the selected grouping algorithm.
rtalign	Set to TRUE to enable retention time alignment.
loadRawData	Set to TRUE if analyses are available as mzXML or mzML files. Otherwise MS data is not loaded, and some dummy data (*e.g.* file paths) is used in the returned object.
groupArgs, alignArgs	Named character vector that may contain extra parameters to be used by KPIC::PICset.group and KPIC::PICset.align, respectively.
verbose	if FALSE then no text output will be shown.

Details

This function uses KPIC2 to group features. This function is called when calling groupFeatures with algorithm="kpic2".

Grouping of features and alignment of their retention times are performed with the KPIC::PICset.group and KPIC::PICset.align functions, respectively.

Value

An object of a class which is derived from featureGroups.

Sets workflows

loadRawData and arguments related to retention time alignment are currently not supported for sets workflows.

References

Ji H, Zeng F, Xu Y, Lu H, Zhang Z (2017). "KPIC2: An Effective Framework for Mass Spectrometry-Based Metabolomics Using Pure Ion Chromatograms." *Analytical Chemistry*, **89**(14), 7631–7640. doi:10.1021/acs.analchem.7b01547.

See Also

groupFeatures for more details and other algorithms.

groupFeaturesOpenMS *Group features using OpenMS*

Description

Group and align features with OpenMS tools

Usage

```
groupFeaturesOpenMS(feat, ...)

## S4 method for signature 'features'
groupFeaturesOpenMS(
  feat,
  rtalign = TRUE,
  QT = FALSE,
  maxAlignRT = 30,
  maxAlignMZ = 0.005,
  maxGroupRT = 12,
  maxGroupMZ = 0.005,
  extraOptsRT = NULL,
  extraOptsGroup = NULL,
  verbose = TRUE
)

## S4 method for signature 'featuresSet'
groupFeaturesOpenMS(feat, ..., verbose = TRUE)
```

Arguments

feat	The features object with the features to be grouped.
...	Further parameters passed to the selected grouping algorithm.
rtalign	Set to TRUE to enable retention time alignment.
QT	If enabled, use FeatureLinkerUnlabeledQT instead of FeatureLinkerUnlabeled for feature grouping.
maxAlignRT, maxAlignMZ	Used for retention alignment. Maximum retention time or m/z difference (seconds/Dalton) for feature pairing. Sets -algorithm:pairfinder:distance_RT:max_difference and -algorithm:pairfinder:distance_MZ:max_difference otpions, respectively.
maxGroupRT, maxGroupMZ	as maxAlignRT and maxAlignMZ, but for grouping of features. Sets -algorithm:distance_RT:max_diff and -algorithm:distance_MZ:max_difference options, respectively.
extraOptsRT, extraOptsGroup	Named list containing extra options that will be passed to MapAlignerPoseClustering or FeatureLinkerUnlabeledQT/FeatureLinkerUnlabeled, respectively. Any

options specified here will override any of the above. Example: `extraOptsGroup=list("-algorithm:di` (corresponds to setting `maxGroupRT=12`). Set to `NULL` to ignore.

`verbose` if `FALSE` then no text output will be shown.

Details

This function uses OpenMS to group features. This function is called when calling `groupFeatures` with `algorithm="openms"`.

Retention times may be aligned by the MapAlignerPoseClustering TOPP tool. Grouping is achieved by either the FeatureLinkerUnlabeled or FeatureLinkerUnlabeledQT TOPP tools.

Value

An object of a class which is derived from `featureGroups`.

References

Rost HL, Sachsenberg T, Aiche S, Bielow C, Weisser H, Aicheler F, Andreotti S, Ehrlich H, Gutenbrunner P, Kenar E, Liang X, Nahnsen S, Nilse L, Pfeuffer J, Rosenberger G, Rurik M, Schmitt U, Veit J, Walzer M, Wojnar D, Wolski WE, Schilling O, Choudhary JS, Malmstrom L, Aebersold R, Reinert K, Kohlbacher O (2016). "OpenMS: a flexible open-source software platform for mass spectrometry data analysis." *Nature Methods*, **13**(9), 741–748. doi:10.1038/nmeth.3959.

pugixml (via Rcpp) is used to process OpenMS XML output.

Eddelbuettel D (2013). *Seamless R and C++ Integration with Rcpp*. Springer, New York. doi:10.1007/9781461468684, ISBN 978-1-4614-6867-7.

Eddelbuettel D, Balamuta J (2018). "Extending R with C++: A Brief Introduction to Rcpp." *The American Statistician*, **72**(1), 28-36. doi:10.1080/00031305.2017.1375990.

Eddelbuettel D, Francois R, Allaire J, Ushey K, Kou Q, Russell N, Ucar I, Bates D, Chambers J (2025). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.0, `https://www.rcpp.org`.

Eddelbuettel D, François R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.1

See Also

`groupFeatures` for more details and other algorithms.

`groupFeaturesSIRIUS` *Group features using SIRIUS*

Description

Uses SIRIUS to find *and* group features.

Usage

```
groupFeaturesSIRIUS(analysisInfo, verbose = TRUE)
```

Arguments

`analysisInfo`	A `data.frame` with Analysis information.
`verbose`	if `FALSE` then no text output will be shown.

Details

This function uses SIRIUS to group features. This function is called when calling `groupFeatures` with `algorithm="sirius"`.

Finding and grouping features is done by running the `lcms-align` command on every analyses at once. For this reason, grouping feature data from other algorithms than `SIRIUS` is not supported.

The MS files should be in the '`mzML`' or '`mzXML`' format. Furthermore, this algorithms requires the presence of (data-dependent) MS/MS data.

The input MS data files need to be centroided. The `convertMSFiles` function can be used to centroid data.

Value

An object of a class which is derived from `featureGroups`.

References

Duhrkop K, Fleischauer M, Ludwig M, Aksenov AA, Melnik AV, Meusel M, Dorrestein PC, Rousu J, Bocker S (2019). "SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information." *Nature Methods*, **16**(4), 299–302. doi:10.1038/s4159201903448.

See Also

`groupFeatures` for more details and other algorithms.

groupFeaturesXCMS *Group features using XCMS (old interface)*

Description

Group and align features with the legacy xcmsSet function from the **xcms** package.

Usage

```
groupFeaturesXCMS(feat, ...)

## S4 method for signature 'features'
groupFeaturesXCMS(
  feat,
  rtalign = TRUE,
  loadRawData = TRUE,
  groupArgs = list(mzwid = 0.015),
  retcorArgs = list(method = "obiwarp"),
  verbose = TRUE
)

## S4 method for signature 'featuresSet'
groupFeaturesXCMS(feat, groupArgs = list(mzwid = 0.015), verbose = TRUE)
```

Arguments

feat	The features object with the features to be grouped.
...	Further parameters passed to the selected grouping algorithm.
rtalign	Set to TRUE to enable retention time alignment.
loadRawData	Set to TRUE if analyses are available as mzXML or mzML files. Otherwise MS data is not loaded, and some dummy data (*e.g.* file paths) is used in the returned object.
groupArgs	named character vector that may contain extra grouping parameters to be used by xcms::group
retcorArgs	named character vector that may contain extra parameters to be used by xcms::retcor.
verbose	if FALSE then no text output will be shown.

Details

This function uses XCMS to group features. This function is called when calling groupFeatures with algorithm="xcms".

Grouping of features and alignment of their retention times are performed with the xcms::group and xcms::retcor functions, respectively. Both functions have an extensive list of parameters to modify their behavior and may therefore be used to potentially optimize results.

Value

An object of a class which is derived from `featureGroups`.

Sets workflows

`loadRawData` and arguments related to retention time alignment are currently not supported for sets workflows.

References

Benton HP, Want EJ, Ebbels TMD (2010). "Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data." *BIOINFORMATICS*, **26**, 2488.

Smith, C.A., Want, E.J., O'Maille, G., Abagyan,R., Siuzdak, G. (2006). "XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification." *Analytical Chemistry*, **78**, 779–787.

Tautenhahn R, Boettcher C, Neumann S (2008). "Highly sensitive feature detection for high resolution LC/MS." *BMC Bioinformatics*, **9**, 504.

See Also

`groupFeatures` for more details and other algorithms.

---

groupFeaturesXCMS3 *Group features using XCMS (new interface)*

Description

Uses the new `xcms3` interface from the **xcms** package to find features.

Usage

```
groupFeaturesXCMS3(feat, ...)

## S4 method for signature 'features'
groupFeaturesXCMS3(
  feat,
  rtalign = TRUE,
  loadRawData = TRUE,
  groupParam = xcms::PeakDensityParam(sampleGroups = analysisInfo(feat)$group),
  preGroupParam = groupParam,
  retAlignParam = xcms::ObiwarpParam(),
  verbose = TRUE
)

## S4 method for signature 'featuresSet'
```

```
groupFeaturesXCMS3(
  feat,
  groupParam = xcms::PeakDensityParam(sampleGroups = analysisInfo(feat)$group),
  verbose = TRUE
)
```

Arguments

feat	The features object with the features to be grouped.
...	Further parameters passed to the selected grouping algorithm.
rtalign	Set to TRUE to enable retention time alignment.
loadRawData	Set to TRUE if analyses are available as mzXML or mzML files. Otherwise MS data is not loaded, and some dummy data (*e.g.* file paths) is used in the returned object.
groupParam, retAlignParam	parameter object that is directly passed to xcms::groupChromPeaks and xcms::adjustRtime, respectively.
preGroupParam	grouping parameters applied when features are grouped *prior* to alignment (only with peak groups alignment).
verbose	if FALSE then no text output will be shown.

Details

This function uses XCMS3 to group features. This function is called when calling groupFeatures with algorithm="xcms3".

Grouping of features and alignment of their retention times are performed with the xcms::groupChromPeaks and xcms::adjustRtime functions, respectively. Both of these functions support an extensive amount of parameters that modify their behavior and may therefore require optimization.

Value

An object of a class which is derived from featureGroups.

Sets workflows

loadRawData and arguments related to retention time alignment are currently not supported for sets workflows.

References

Benton HP, Want EJ, Ebbels TMD (2010). "Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data." *BIOINFORMATICS*, **26**, 2488.

Smith, C.A., Want, E.J., O'Maille, G., Abagyan,R., Siuzdak, G. (2006). "XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification." *Analytical Chemistry*, **78**, 779–787.

Tautenhahn R, Boettcher C, Neumann S (2008). "Highly sensitive feature detection for high resolution LC/MS." *BMC Bioinformatics*, **9**, 504.

See Also

groupFeatures for more details and other algorithms.

---

groupTable *Base class for grouped features.*

Description

This class holds all the information for grouped features.

Usage

```
groupTable(object, ...)

groupFeatIndex(fGroups)

groupInfo(fGroups)

unique(x, incomparables = FALSE, ...)

overlap(fGroups, which, exclusive = FALSE, ...)

selectIons(fGroups, components, prefAdduct, ...)

groupQualities(fGroups)

groupScores(fGroups)

internalStandards(fGroups)

internalStandardAssignments(fGroups, ...)

normInts(
  fGroups,
  featNorm = "none",
  groupNorm = FALSE,
  normFunc = max,
  standards = NULL,
  ISTDRTWindow = 120,
  ISTDMZWindow = 300,
  minISTDs = 3,
  ...
)

## S4 method for signature 'featureGroups'
names(x)
```

```
## S4 method for signature 'featureGroups'
analyses(obj)

## S4 method for signature 'featureGroups'
replicateGroups(obj)

## S4 method for signature 'featureGroups'
groupNames(obj)

## S4 method for signature 'featureGroups'
length(x)

## S4 method for signature 'featureGroups'
show(object)

## S4 method for signature 'featureGroups'
groupTable(object, areas = FALSE, normalized = FALSE)

## S4 method for signature 'featureGroups'
analysisInfo(obj)

## S4 method for signature 'featureGroups'
groupInfo(fGroups)

## S4 method for signature 'featureGroups'
featureTable(obj)

## S4 method for signature 'featureGroups'
getFeatures(obj)

## S4 method for signature 'featureGroups'
groupFeatIndex(fGroups)

## S4 method for signature 'featureGroups'
groupQualities(fGroups)

## S4 method for signature 'featureGroups'
groupScores(fGroups)

## S4 method for signature 'featureGroups'
annotations(obj)

## S4 method for signature 'featureGroups'
internalStandards(fGroups)

## S4 method for signature 'featureGroups'
internalStandardAssignments(fGroups)
```

```
## S4 method for signature 'featureGroups'
adducts(obj)

## S4 replacement method for signature 'featureGroups'
adducts(obj) <- value

## S4 method for signature 'featureGroups'
concentrations(fGroups)

## S4 method for signature 'featureGroups'
toxicities(fGroups)

## S4 method for signature 'featureGroups,ANY,ANY,missing'
x[i, j, ..., rGroups, results, drop = TRUE]

## S4 method for signature 'featureGroups,ANY,ANY'
x[[i, j]]

## S4 method for signature 'featureGroups'
x$name

## S4 method for signature 'featureGroups'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featureGroups'
export(obj, type, out)

## S4 method for signature 'featureGroups'
as.data.table(
  x,
  average = FALSE,
  areas = FALSE,
  features = FALSE,
  qualities = FALSE,
  regression = FALSE,
  averageFunc = mean,
  normalized = FALSE,
  FCParams = NULL,
  concAggrParams = getDefPredAggrParams(),
  toxAggrParams = getDefPredAggrParams(),
  normConcToTox = FALSE
)

## S4 method for signature 'featureGroups'
unique(x, which, relativeTo = NULL, outer = FALSE)

## S4 method for signature 'featureGroups'
```

```
overlap(fGroups, which, exclusive)

## S4 method for signature 'featureGroups'
calculatePeakQualities(
  obj,
  weights,
  flatnessFactor,
  avgFunc = mean,
  parallel = TRUE
)

## S4 method for signature 'featureGroups'
selectIons(
  fGroups,
  components,
  prefAdduct,
  onlyMonoIso = TRUE,
  chargeMismatch = "adduct"
)

## S4 method for signature 'featureGroups'
normInts(
  fGroups,
  featNorm = "none",
  groupNorm = FALSE,
  normFunc = max,
  standards = NULL,
  ISTDRTWindow = 120,
  ISTDMZWindow = 300,
  minISTDs = 3,
  ...
)

## S4 method for signature 'featureGroups'
getTICs(obj, retentionRange = NULL, MSLevel = c(1, 2))

## S4 method for signature 'featureGroups'
getBPCs(obj, retentionRange = NULL, MSLevel = c(1, 2))

## S4 method for signature 'featureGroupsSet'
sets(obj)

## S4 method for signature 'featureGroupsSet'
internalStandardAssignments(fGroups, set = NULL)

## S4 method for signature 'featureGroupsSet'
adducts(obj, set, ...)
```

```
## S4 replacement method for signature 'featureGroupsSet'
adducts(obj, set, reGroup = TRUE) <- value

## S4 method for signature 'featureGroupsSet'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featureGroupsSet'
show(object)

## S4 method for signature 'featureGroupsSet'
featureTable(obj)

## S4 method for signature 'featureGroupsSet,ANY,ANY,missing'
x[i, j, ..., rGroups, sets = NULL, drop = TRUE]

## S4 method for signature 'featureGroupsSet'
export(obj, type, out, set)

## S4 method for signature 'featureGroupsSet'
unique(x, which, ..., sets = FALSE)

## S4 method for signature 'featureGroupsSet'
overlap(fGroups, which, exclusive, sets = FALSE)

## S4 method for signature 'featureGroupsSet'
selectIons(fGroups, components, prefAdduct, ...)

## S4 method for signature 'featureGroupsSet'
normInts(
  fGroups,
  featNorm = "none",
  groupNorm = FALSE,
  normFunc = max,
  standards = NULL,
  ISTDRTWindow = 120,
  ISTDMZWindow = 300,
  minISTDs = 3,
  ...
)

## S4 method for signature 'featureGroupsSet'
unset(obj, set)

## S4 method for signature 'featureGroupsKPIC2'
delete(obj, ...)

## S4 method for signature 'featureGroups'
plotTICs(
```

```
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  colourBy = c("none", "analyses", "rGroups"),
  showLegend = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'featureGroups'
plotBPCs(
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  colourBy = c("none", "analyses", "rGroups"),
  showLegend = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'featureGroupsXCMS'
delete(obj, ...)

## S4 method for signature 'featureGroupsXCMS3'
delete(obj, ...)
```

Arguments

`...`	For the `"["` operator: ignored. For `delete`: passed to the function specified as `j`. For `normInts`: passed to `screenSuspects` if `featNorm="istd"`. For sets workflow methods: further arguments passed to the base `featureGroups` method.
`fGroups, obj, x, object`	`featureGroups` object to be accessed.
`incomparables`	Ignored.
`which`	A character vector with replicate groups used for comparison. For `overlap`: can also be a `list` of `character` vectors with replicate groups to compare. For instance, `which=list(c("samp1", "samp2"), c("samp3", "samp4"))` returns the overlap between `"samp1"+"samp2"` and `"samp3"+"samp4"`.
`exclusive`	If `TRUE` then all feature groups are removed that are not unique to the given replicate groups.

components — The components object that was generated for the given featureGroups object. Obviously, the components must be created with algorithms that support adduct/isotope annotations, such as those from **RAMClustR** and **cliqueMS**.

prefAdduct — The 'preferred adduct' (see method description). This is often "[M+H]+" or "[M-H]-".

featNorm — The method applied for feature normalization: "istd", "tic", "conc" or "none". See the Feature intensity normalization section for details.

groupNorm — If TRUE then group normalization is performed. See the Feature intensity normalization section for details.

normFunc — A function to combine data for normalization. See the Feature intensity normalization section for details.

standards — A data.table (or data.frame) with all internal standards. Should follow the format of a suspect list. Only used if featNorm="istd". See the Feature intensity normalization section for details.

(**sets workflow**) Can also be a list with internal standard lists.

See the suspects argument to screenSuspects for more details.

ISTDRTWindow, ISTDMZWindow — The retention time and *m/z* windows for IS selection. Only used if featNorm="istd". See the Feature intensity normalization section for details.

minISTDs — The minimum number of IS that should be assigned to each feature (if possible). Only used if featNorm="istd". See the Feature intensity normalization section for details.

areas — If set to TRUE then areas are considered instead of peak intensities.

For as.data.table: ignored if features=TRUE, as areas of features are always reported.

normalized — If TRUE then normalized intensity data is used (see the Feature intensity normalization section.

For as.data.table: if no normalization data is available (*e.g.* because normInts was not used) then an automatic group normalization is performed.

value — For adducts<-: A character with adduct annotations assigned to each feature group. The length should equal the number of feature groups. Can be named with feature group names to customize the assignment order.

i, j — For [/[[: A numeric or character value which is used to select analyses/feature groups by their index or name, respectively (for the order/names see analyses()/names()).

For [: Can also be logical to perform logical selection (similar to regular vectors). If missing all analyses/feature groups are selected.

For [[: should be a scalar value. If j is not specified, i selects by feature groups instead.

For delete: The data to remove from. i are the analyses as numeric index, logical or character, j the feature groups as numeric index, logical or character. If either is NULL then data for all is removed. j may also be a function: it will be called for each feature group, with a vector of the group intensities, the group

name and any other arguments passed as ... to delete. The return value of this function specifies the analyses of the features in the group to be removed (same format as i).

rGroups — For [: An optional character vector: if specified only keep results for the given replicate groups (equivalent to the rGroups argument to filter).

results — Optional argument. If specified only feature groups with results in the specified object are kept. The class of results should be featureAnnotations or components. Multiple objects can be specified in a list: in this case a feature group is kept if it has a result in *any* of the objects (equivalent to the results argument to filter).

drop — ignored.

name — The feature group name (partially matched).

type — The export type: "brukerpa" (Bruker ProfileAnalysis), "brukertasq" (Bruker TASQ) or "mzmine" (MZmine).

out — The destination file for the exported data.

average — If TRUE then data within replicate groups are averaged.

For as.data.table: if features=TRUE other feature properties are also averaged.

features — If TRUE then feature specific data will be added. If average=TRUE this data will be averaged for each feature group.

qualities — Adds feature (group) qualities (qualities="quality"), scores (qualities="score") or both (qualities="both"), if this data is available (*i.e.* from calculatePeakQualities). If qualities=FALSE then nothing is reported.

regression — Set to TRUE to add regression data for each feature group. For this a linear model is created (intensity/area [depending on areas argument] *vs* concentration). The model concentrations (e.g. of a set of standards) is derived from the conc column of the analysis information. From this model the intercept, slope and R2 is added to the output. In addition, when features=TRUE, concentrations for each feature are added. Note that no regression information is added when no conc column is present in the analysis information or when less than two concentrations are specified (*i.e.* the minimum amount).

averageFunc — Function used for averaging. Only used when average=TRUE or FCParams != NULL.

FCParams — A parameter list to calculate Fold change data. See getFCParams for more details. Set to NULL to not perform FC calculations.

concAggrParams, toxAggrParams — Parameters to aggregate calculated concentrations/toxicities (obtained with calculateConcs/calculateT
See prediction aggregation parameters for more information. Set to NULL to omit this data.

normConcToTox — Set to TRUE to normalize concentrations to toxicities. Only relevant if this data is present (see calculateConcs/calculateTox).

relativeTo — A character vector with replicate groups that should be used for unique comparison. If NULL then all replicate groups are used for comparison. Replicate groups specified in which are ignored.

outer	If TRUE then only feature groups are kept which do not overlap between the specified replicate groups for the which parameter.
weights	A named numeric vector that defines the weight for each score to calculate the totalScore. The names of the vector follow the score names. Unspecified weights are defaulted to '1'. Example: weights=c(ApexBoundaryRatioScore=0.5, GaussianSimilarityScore=2).
flatnessFactor	Passed to **MetaClean** as the flatness.factor argument to calculateJaggedness and calculateModality.
avgFunc	The function used to average the peak qualities and scores for each feature group.
parallel	If set to TRUE then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.
onlyMonoIso	Set to TRUE to only keep feature groups that were annotated as monoisotopic. Feature groups are never removed by this setting if no isotope annotations are available.
chargeMismatch	Specifies how to deal with a mismatch in charge between adduct and isotope annotations. Valid values are: "adduct" (ignore isotope annotation), "isotope" (ignore adduct annotation), "none" (ignore both annotations) and "ignore" (don't check for charge mismatches). *Important*: when OpenMS is used to find features, it already removes any detected non-monoisotopic features by default. Hence, in such case setting chargeMismatch="adduct" is more appropriate.
retentionRange	Range of retention time (in seconds) to collect TIC traces. Should be a numeric vector with length of two containing the min/max values. Set to NULL to ignore.
MSLevel	Integer vector with the ms levels (i.e., 1 for MS1 and 2 for MS2) to obtain TIC traces.
set	(**sets workflow**) The name of the set.
reGroup	(**sets workflow**) Set to TRUE to re-group the features after the adduct annotations are changed. See the Sets workflow section for more details.
sets	(**sets workflow**) For [: a character with name(s) of the sets to keep. For overlap and unique: If TRUE then the which argument changes its meaning and is used to specify the names of the sets to be compared.
retMin	Plot retention time in minutes (instead of seconds).
title	Character string used for title of the plot. If NULL a title will be automatically generated.
colourBy	Sets the automatic colour selection: "none" for a single colour or "analyses"/"rGroups" for a distinct colour per analysis or analysis replicate group.
showLegend	Plot a legend if TRUE.
xlim, ylim	Sets the plot size limits used by plot. Set to NULL for automatic plot sizing.

Details

The featureGroup class is the workhorse of **patRoon**: almost all functionality operate on its instantiated objects. The class holds all information from grouped features (obtained from features). This class itself is virtual, hence, objects are not created directly from it. Instead, 'feature groupers' such as groupFeaturesXCMS return a featureGroups derived object after performing the actual grouping of features across analyses.

Value

`delete` returns the object for which the specified data was removed.

`calculatePeakQualities` returns a modified object amended with peak qualities and scores.

`selectIons` returns a `featureGroups` object with only the selected feature groups and amended with adduct annotations.

`normInts` returns a `featureGroups` object, amended with data in the `ISTDs` and `ISTDAssignments` slots if `featNorm="istd"`.

Methods (by generic)

- `names(featureGroups)`: Obtain feature group names.
- `analyses(featureGroups)`: returns a `character` vector with the names of the analyses for which data is present in this object.
- `replicateGroups(featureGroups)`: returns a `character` vector with the names of the replicate groups for which data is present in this object.
- `groupNames(featureGroups)`: Same as `names`. Provided for consistency to other classes.
- `length(featureGroups)`: Obtain number of feature groups.
- `show(featureGroups)`: Shows summary information for this object.
- `groupTable(featureGroups)`: Accessor for `groups` slot.
- `analysisInfo(featureGroups)`: Obtain analysisInfo (see analysisInfo slot in `features`).
- `groupInfo(featureGroups)`: Accessor for `groupInfo` slot.
- `featureTable(featureGroups)`: Obtain feature information (see `features`).
- `getFeatures(featureGroups)`: Accessor for `features` slot.
- `groupFeatIndex(featureGroups)`: Accessor for `ftindex` slot.
- `groupQualities(featureGroups)`: Accessor for `groupQualities` slot.
- `groupScores(featureGroups)`: Accessor for `groupScores` slot.
- `annotations(featureGroups)`: Accessor for `annotations` slot.
- `internalStandards(featureGroups)`: Accessor for `ISTDs` slot.
- `internalStandardAssignments(featureGroups)`: Accessor for `ISTDAssignments` slot.
- `adducts(featureGroups)`: Returns a named `character` with adduct annotations assigned to each feature group (if available).
- `adducts(featureGroups) <- value`: Sets adduct annotations for feature groups.
- `concentrations(featureGroups)`: Accessor for `concentrations` slot.
- `toxicities(featureGroups)`: Accessor for `toxicities` slot.
- `x[i`: Subset on analyses/feature groups.
- `x[[i`: Extract intensity values.
- `$`: Extract intensity values for a feature group.
- `delete(featureGroups)`: Completely deletes specified feature groups.

- `export(featureGroups)`: Exports feature groups to a '`.csv`' file that is readable to Bruker ProfileAnalysis (a 'bucket table'), Bruker TASQ (an analyte database) or that is suitable as input for the `Targeted peak detection` functionality of MZmine.
- `as.data.table(featureGroups)`: Obtain a summary table (a `data.table`) with retention, *m/z*, intensity and optionally other feature data.
- `unique(featureGroups)`: Obtain a subset with unique feature groups present in one or more specified replicate group(s).
- `overlap(featureGroups)`: Obtain a subset with feature groups that overlap between a set of specified replicate group(s).
- `calculatePeakQualities(featureGroups)`: Calculates peak and group qualities for all features and feature groups. The peak qualities (and scores) are calculated with the features method of this function, and subsequently averaged per feature group. Then, **MetaClean** is used to calculate the `Elution Shift` and `Retention Time Consistency` group quality metrics (see the **MetaClean** publication cited below for more details). Similarly to the `features` method, these metrics are scored by normalizing qualities among all groups and scaling them from '`0`' (worst) to '`1`' (best). The `totalScore` for each group is then calculated as the weighted sum from all feature (group) scores. The `getMCTrainData` and `predictCheckFeaturesSession` functions can be used to train and apply Pass/Fail ML models from **MetaClean**.
- `selectIons(featureGroups)`: uses componentization results to select feature groups with preferred adduct ion and/or isotope annotation. Typically, this means that only feature groups are kept if they are (de-)protonated adducts and are monoisotopic. The adduct annotation assignments for the selected feature groups are copied from the components to the `annotations` slot. If the adduct for a feature group is unknown, its annotation is defaulted to the 'preferred' adduct, and hence, the feature group will never be removed. Furthermore, if a component does not contain an annotation with the preferred adduct, the most intense feature group is selected instead. Similarly, if no isotope annotation is available, the feature group is assumed to be monoisotopic and thus not removed. An important advantage of `selectIons` is that it may considerably simplify your dataset. Furthermore, the adduct assignments allow formula/compound annotation steps later in the workflow to improve their annotation accuracy. On the other hand, it is important the componentization results are reliable. Hence, it is highly recommended that, prior to calling `selectIons`, the settings to `generateComponents` are optimized and its results are reviewed with `checkComponents`. Finally, the `adducts<-` method can be used to manually correct adduct assignments afterwards if necessary.
- `normInts(featureGroups)`: Provides various methods to normalizes feature intensities for each sample analysis or of all features within a feature group. See the `Feature intensity normalization` section below.
- `getTICs(featureGroups)`: Obtain the total ion chromatogram/s (TICs) of the analyses.
- `getBPCs(featureGroups)`: Obtain the base peak chromatogram/s (BPCs) of the analyses.
- `plotTICs(featureGroups)`: Plots the total ion chromatogram/s (TICs) of the analyses.
- `plotBPCs(featureGroups)`: Plots the base peak chromatogram/s (BPCs) of the analyses.

Slots

`groups` Matrix (`data.table`) with intensities for each feature group (columns) per analysis (rows). Access with `groups` method.

`analysisInfo,features` Analysis info and `features` class associated with this object. Access with `analysisInfo` and `featureTable` methods, respectively.

`groupInfo` `data.frame` with retention time (`rts` column, in seconds) and *m/z* (`mzs` column) for each feature group. Access with `groupInfo` method.

`ftindex` Matrix (`data.table`) with feature indices for each feature group (columns) per analysis (rows). Each index corresponds to the row within the feature table of the analysis (see `featureTable`).

`groupQualities,groupScores` A `data.table` with qualities/scores for each feature group (see the `calculatePeakQualities` method).

`annotations` A `data.table` with adduct annotations for each group (see the `selectIons` method).

`ISTDs` A `data.table` with screening results for internal standards (filled in by the `normInts` method).

`ISTDAssignments` A `list`, where each item is named by a feature group and consists of a vector with feature group names of the internal standards assigned to it (filled in by the `normInts` method).

`concentrations,toxicities` A `data.table` with predicted concentrations/toxicities for each feature group. Assigned by the `calculateConcs`/`calculateTox` methods. Use the `concentratrions/toxicities` methods for access.

`groupAlgo,groupArgs,groupVerbose` (**sets workflow**) Grouping parameters that were used when this object was created. Used by `adducts<-` and `selectIons` when these methods perform a re-grouping of features.

`annotations,ISTDAssignments` (**sets workflow**) As the `featureGroups` slots, but contains the data per set.

`annotationsChanged` Set internally by `adducts()<-` and applied as soon as `reGroup=TRUE`.

S4 class hierarchy

- `workflowStep`
  - `featureGroups`
    - `featureGroupsSet`
      - `featureGroupsScreeningSet`
    - `featureGroupsUnset`
    - `featureGroupsScreening`
      - `featureGroupsSetScreeningUnset`
    - `featureGroupsBruker`
    - `featureGroupsConsensus`
    - `featureGroupsEnviMass`
    - `featureGroupsKPIC2`
    - `featureGroupsOpenMS`
    - `featureGroupsSIRIUS`
    - `featureGroupsBrukerTASQ`
    - `featureGroupsXCMS`
    - `featureGroupsXCMS3`

Feature intensity normalization

The normInts method performs normalization of feature intensities (and areas). These values are amended in the features slot, while the original intensities/areas are kept. To use the normalized intensities set normalized=TRUE to methods such as plotInt, generateComponentsIntClust and as.data.table. Please see the normalized argument documentation for these methods for more details.

The normInts method supports several methods to normalize intensities/areas of features within the same analysis. Most methods are influenced by the *normalization concentration* (norm_conc in the analysis information) set for each sample analysis. For NA or zero values the output will be zero. If norm_conc is completely absent from the analysis information or all values are NA, the normalization concentration is defaulted to one.

The different normalization methods are:

1. featNorm="istd" Uses *internal standards* (IS) for normalization. The IS are screened internally by the screenSuspects function. Hence, the IS specified by the standards argument should follow the format of a suspect list. Note that labelled elements in IS formulae should be specified with the **rcdk** format, *e.g.* "[13]C" for 13C, "[2]H" for a deuterium etc. Example IS lists are provided with the **patRoonData** package.

   The assignment of IS to features is automatically performed, using the following criteria:

   (a) Only analyses are considered with a defined normalization concentration.
   (b) The IS must be detected in all of the analyses in which the feature was detected.
   (c) The retention time and *m/z* are reasonably close (ISTDRTWindow/ISTDMZWindow arguments). However, additional IS candidates outside these windows will be chosen if the number of candidates is less than the minISTDs argument. In this case the next close(st) candidate(s) will be chosen.

   Normalization of features within the same feature group always occur with the same IS. If multiple IS are assigned to a feature then normalization occurs with the combined intensity (area), which is calculated with the function defined by the normFunc argument. The (combined) IS intensity is then normalized by the normalization concentration, and finally used for feature normalization.

2. featNorm="tic" Uses the Total Ion Current (TIC) to normalize intensities. The TIC is calculated by combining all intensities with the function defined by the normFunc argument. For this reason, you may need to take care to perform normalization before *e.g.* suspect screening or other prioritization techniques. The TIC normalized intensities are finally divided by the normalization concentration.

3. featNorm="conc" Simply divides all intensities (areas) with the normalization concentration defined for the sample.

4. featNorm="none" Performs no normalization. The raw intensity values are simply copied. This is mainly useful if you only want to do group normalization (described below).

The meaning of the normalization concentration differs for each method: for "istd" it resembles the IS concentration of a sample analysis, whereas for "tic" and "conc" it is used to normalize different sample amounts (*e.g.* injection volume).

If groupNorm=TRUE then feature intensities (areas) will be normalized by the combined values for its feature group (again, combination occurs with normFunc). This *group normalization* always

occurs *after* aforementioned normalization methods. Group normalization was the only method with **patRoon** '<2.1', and still occurs automatically if `normInts` was not called when a method is executed that requests normalized data.

Sets workflows

The `featureGroupsSet` class is applicable for sets workflows. This class is derived from `featureGroups` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- `sets` Returns the set names for this object.
- `unset` Converts the object data for a specified set into a 'non-set' object (`featureGroupsUnset`), which allows it to be used in 'regular' workflows. The adduct annotations for the selected set are used to convert all feature (group) masses to ionic *m/z* values. The annotations persist in the converted object.

The following methods are changed or with new functionality:

- `adducts`, `adducts<-` require the `set` argument. The order of the data that is returned/changed follows that of the `annotations` slot. Furthermore, `adducts<-` will perform a re-grouping of features when its `reGroup` parameter is set to `TRUE`. The implications for this are discussed below. Note that no adducts are changed *until* `reGroup=TRUE`.
- the subset operator (`[`) has specific arguments to choose (feature presence in) sets. See the argument descriptions.
- `as.data.table`: normalization of intensities is performed per set.
- `export` Only allows to export data from one set. The `unset` method is used prior to exporting the data.
- `overlap` and `unique` allow to handle data per set. See the `sets` argument description.
- `selectIons` Will perform a re-grouping of features. The implications of this are discussed below.
- `normInts` Performs normalization for each set *independently*.

A re-grouping of features occurs if `selectIons` is called or `adducts<-` is used with `reGroup=TRUE`. Afterwards, it is very likely that feature group names are changed. Since data generated later in the workflow (*e.g.* annotation steps) rely on feature group names, these objects are **not valid** anymore, and **must** be re-generated.

Author(s)

Rick Helmus <<r.helmus@uva.nl>> and Ricardo Cunha <<cunha@iuta.de>> (`getTICs` and `getBPCs` functions)

References

Chetnik K, Petrick L, Pandey G (2020). "MetaClean: a machine learning-based classifier for reduced false positive peak detection in untargeted LC-MS metabolomics data." *Metabolomics*, **16**(11). doi:10.1007/s11306020017383.

See Also

groupFeatures for generating feature groups, feature-filtering and feature-plotting for more advanced featureGroups methods.

---

importFeatureGroups *Import feature groups from files*

Description

Generic function to import feature groups produced by other software from files.

Usage

```
importFeatureGroups(path, type, ...)
```

Arguments

path	The path that should be used for importing. See the algorithm specific functions for more details.
type	Which file type should be imported: "brukerpa" (Bruker ProfileAnalysis), "brukertasq" (Bruker TASQ) or "envimass" (**enviMass**).
...	Further arguments passed to the selected import algorithm function.

Details

importFeatureGroups is a generic function that will import feature groups from files by one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as importFeatureGroupsBrukerTASQ and importFeatureGroupsBrukerPA. While these functions may be called directly, importFeatureGroups provides a generic interface and is therefore usually preferred.

Value

An object of a class which is derived from featureGroups.

See Also

The featureGroups output class and its methods and the algorithm specific functions: importFeatureGroupsBrukerPA, importFeatureGroupsBrukerTASQ, importFeatureGroupsEnviMass

groupFeatures to group features. Other import functions: importFeatureGroupsXCMS, importFeatureGroupsXCMS3 and importFeatureGroupsKPIC2.

`importFeatureGroupsBrukerPA` *Imports feature groups from Bruker ProfileAnalysis*

Description

Imports a 'bucket table' produced by Bruker ProfileAnalysis (PA)

Usage

```
importFeatureGroupsBrukerPA(
  path,
  feat,
  rtWindow = 12,
  mzWindow = 0.005,
  intWindow = 5,
  warn = TRUE
)
```

Arguments

`path`	The file path to a exported 'bucket table' '`.txt`' file from PA.
`feat`	The `features` object obtained with `findFeaturesBruker`.
`rtWindow`, `mzWindow`, `intWindow`	Search window values for retention time (seconds), *m/z* (Da) and intensity used to find back features within feature groups from PA (+/- the retention/mass/intensity value of a feature).
`warn`	Warn about missing or duplicate features when relating them back from grouped features.

Details

This function imports data from Bruker ProfileAnalysis. This function is called when calling `importFeatureGroups` with `type="brukerpa"`.

The 'bucket table' should be exported as '`.txt`' file. Please note that this function only supports features generated by `findFeaturesBruker` and it is **crucial** that DataAnalysis files remain unchanged when features are collected and the bucket table is generated. Furthermore, please note that PA does not retain information about originating features for generated buckets. For this reason, this function tries to find back the original features and care must be taken to correctly specify search parameters (`rtWindow`, `mzWindow`, `intWindow`).

Value

An object of a class which is derived from `featureGroups`.

See Also

`importFeatureGroups` for more details and other algorithms.

`importFeatureGroupsBrukerTASQ`

Imports feature groups from Bruker TASQ

Description

Imports screening results from Bruker TASQ as feature groups.

Usage

```
importFeatureGroupsBrukerTASQ(path, analysisInfo, clusterRTWindow = 12)
```

Arguments

`path`	The file path to an Excel export of the Global results table from TASQ, converted to '`.csv`' format.
`analysisInfo`	A `data.frame` with Analysis information.
`clusterRTWindow`	This retention time window (in seconds) is used to group hits across analyses together. See also the details section.

Details

This function imports data from Bruker TASQ. This function is called when calling `importFeatureGroups` with `type="brukertasq"`.

The feature groups across analyses are formed based on the name of suspects and their closeness in retention time. The latter is necessary because TASQ does not necessarily perform checks on retention times and may therefore assign a suspect to peaks with different retention times across analyses (or within a single analysis). Hence, suspects with equal names are hierarchically clustered on their retention times (using **fastcluster**) to form the feature groups. The cut-off value for this is specified by the `clusterRTWindow` argument. The input for this function is obtained by generating an Excel export of the 'global' results and subsequently converting the file to '`.csv`' format.

Value

A new `featureGroups` object containing converted screening results from Bruker TASQ.

Note

This function uses estimated min/max values for retention times and dummy min/max *m/z* values for conversion to features, since this information is not (readily) available. Hence, when plotting, for instance, extracted ion chromatograms (with `plotChroms`) the integrated chromatographic peak range shown is incorrect.

This function may use suspect names to base file names used for reporting, logging etc. Therefore, it is important that these are file-compatible names.

References

Müllner D (2013). "fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python." *Journal of Statistical Software*, **53**(9), 1–18. doi:10.18637/jss.v053.i09.

See Also

`importFeatureGroups` for more details and other algorithms.

`importFeatureGroupsEnviMass`

Imports feature groups from enviMass

Description

Imports a 'profiles' produced by **enviMass**.

Usage

```
importFeatureGroupsEnviMass(path, feat, positive)
```

Arguments

`path`	The path of the enviMass project.
`feat`	The `features` object obtained with `importFeaturesEnviMass`.
`positive`	Whether data from positive (`TRUE`) or negative (`FALSE`) should be loaded.

Details

This function imports data from enviMass. This function is called when calling `importFeatureGroups` with `type="envimass"`.

This function *only* imports 'raw' profiles, *not* any results from further componentization steps performed in **enviMass**. Furthermore, this functionality has only been tested with older versions of **enviMass**. Finally, please note that this function only supports features imported by `importFeaturesEnviMass` (obviously, the same project should be used for both importing functions).

Value

An object of a class which is derived from `featureGroups`.

See Also

`importFeatureGroups` for more details and other algorithms.

importFeatureGroupsKPIC2

Imports feature groups from KPIC2

Description

Imports grouped features from an **KPIC** object.

Usage

```
importFeatureGroupsKPIC2(picsSetGrouped, analysisInfo)
```

Arguments

picsSetGrouped	A grouped PIC set object (*e.g.* as returned by KPIC::PICset.group).
analysisInfo	A data.frame with Analysis information.

Value

An object of a class which is derived from featureGroups.

References

Ji H, Zeng F, Xu Y, Lu H, Zhang Z (2017). "KPIC2: An Effective Framework for Mass Spectrometry-Based Metabolomics Using Pure Ion Chromatograms." *Analytical Chemistry*, **89**(14), 7631–7640. doi:10.1021/acs.analchem.7b01547.

See Also

groupFeatures

importFeatureGroupsXCMS

Imports feature groups from XCMS (old interface)

Description

Imports grouped features from a legacy xcmsSet object from the **xcms** package.

Usage

```
importFeatureGroupsXCMS(xs, analysisInfo)
```

Arguments

xs	An xcmsSet object.
analysisInfo	A data.frame with Analysis information.

Value

An object of a class which is derived from featureGroups.

References

Benton HP, Want EJ, Ebbels TMD (2010). "Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data." *BIOINFORMATICS*, **26**, 2488.

Smith, C.A., Want, E.J., O'Maille, G., Abagyan,R., Siuzdak, G. (2006). "XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification." *Analytical Chemistry*, **78**, 779–787.

Tautenhahn R, Boettcher C, Neumann S (2008). "Highly sensitive feature detection for high resolution LC/MS." *BMC Bioinformatics*, **9**, 504.

See Also

importFeaturesXCMS3 and groupFeatures

---

importFeatureGroupsXCMS3

Imports feature groups from XCMS (new interface)

Description

Imports grouped features from a XCMSnExp object from the **xcms** package.

Usage

```
importFeatureGroupsXCMS3(xdata, analysisInfo)
```

Arguments

xdata	An XCMSnExp object.
analysisInfo	A data.frame with Analysis information.

Value

An object of a class which is derived from featureGroups.

References

Benton HP, Want EJ, Ebbels TMD (2010). "Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data." *BIOINFORMATICS*, **26**, 2488.

Smith, C.A., Want, E.J., O'Maille, G., Abagyan,R., Siuzdak, G. (2006). "XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification." *Analytical Chemistry*, **78**, 779–787.

Tautenhahn R, Boettcher C, Neumann S (2008). "Highly sensitive feature detection for high resolution LC/MS." *BMC Bioinformatics*, **9**, 504.

See Also

`groupFeatures`

`importFeatures` *Import features*

Description

Generic function to import features produced by other software.

Usage

```
importFeatures(analysisInfo, type, ...)
```

Arguments

`analysisInfo`	A `data.frame` with Analysis information.
`type`	What type of data should be imported: `"xcms"`, `"xcms3"`, `"kpic2"` or `"envimass"`.
`...`	Further arguments passed to the selected import algorithm function.

Details

`importFeatures` is a generic function that will import features by one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as `importFeaturesXCMS3` and `importFeaturesKPIC2`. While these functions may be called directly, `importFeatures` provides a generic interface and is therefore usually preferred.

Value

An object of a class which is derived from `features`.

See Also

The `features` output class and its methods and the algorithm specific functions: `importFeaturesXCMS`, `importFeaturesXCMS3`, `importFeaturesKPIC2`, `importFeaturesEnviMass`

`findFeatures` to find new features.

importFeaturesEnviMass *Imports features from enviMass*

Description

Imports features from a project generated by the **enviMass** package.

Usage

```
importFeaturesEnviMass(analysisInfo, enviProjPath)
```

Arguments

analysisInfo	A data.frame with Analysis information.
enviProjPath	The path of the enviMass project.

Details

This function imports data from enviMass. This function is called when calling importFeatures with type="envimass".

Value

An object of a class which is derived from features.

Note

This functionality has only been tested with older versions of **enviMass**.

See Also

importFeatures for more details and other algorithms.

importFeaturesKPIC2 *Imports features from KPIC2*

Description

Imports feature data generated by the **KPIC2** package.

Usage

```
importFeaturesKPIC2(picsList, analysisInfo)
```

Arguments

picsList	A list with a pics objects obtained with getPIC or getPIC.kmeans for each analysis.
analysisInfo	A data.frame with Analysis information.

Details

This function imports data from KPIC2. This function is called when calling importFeatures with type="kpic2".

Value

An object of a class which is derived from features.

References

Ji H, Zeng F, Xu Y, Lu H, Zhang Z (2017). "KPIC2: An Effective Framework for Mass Spectrometry-Based Metabolomics Using Pure Ion Chromatograms." *Analytical Chemistry*, **89**(14), 7631–7640. doi:10.1021/acs.analchem.7b01547.

See Also

importFeatures for more details and other algorithms.

---

importFeaturesXCMS *Imports features from XCMS (old interface)*

Description

Imports feature data generated with the legacy xcmsSet function from the **xcms** package.

Usage

```
importFeaturesXCMS(xs, analysisInfo)
```

Arguments

xs	An xcmsSet object.
analysisInfo	A data.frame with Analysis information.

Details

This function imports data from XCMS. This function is called when calling importFeatures with type="xcms".

Value

An object of a class which is derived from features.

References

Benton HP, Want EJ, Ebbels TMD (2010). "Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data." *BIOINFORMATICS*, **26**, 2488.

Smith, C.A., Want, E.J., O'Maille, G., Abagyan,R., Siuzdak, G. (2006). "XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification." *Analytical Chemistry*, **78**, 779–787.

Tautenhahn R, Boettcher C, Neumann S (2008). "Highly sensitive feature detection for high resolution LC/MS." *BMC Bioinformatics*, **9**, 504.

See Also

`importFeatures` for more details and other algorithms.

`importFeaturesXCMS3`

`importFeaturesXCMS3` *Imports features from XCMS (new interface)*

Description

Imports feature data generated from an existing `XCMSnExp` object generated by the **xcms** package.

Usage

```
importFeaturesXCMS3(xdata, analysisInfo)
```

Arguments

`xdata`	An `XCMSnExp` object.
`analysisInfo`	A `data.frame` with Analysis information.

Details

This function imports data from XCMS3. This function is called when calling `importFeatures` with `type="xcms3"`.

Value

An object of a class which is derived from `features`.

References

Benton HP, Want EJ, Ebbels TMD (2010). "Correction of mass calibration gaps in liquid chromatography-mass spectrometry metabolomics data." *BIOINFORMATICS*, **26**, 2488.

Smith, C.A., Want, E.J., O'Maille, G., Abagyan,R., Siuzdak, G. (2006). "XCMS: Processing mass spectrometry data for metabolite profiling using nonlinear peak alignment, matching and identification." *Analytical Chemistry*, **78**, 779–787.

Tautenhahn R, Boettcher C, Neumann S (2008). "Highly sensitive feature detection for high resolution LC/MS." *BMC Bioinformatics*, **9**, 504.

See Also

`importFeatures` for more details and other algorithms.

`loadMSLibrary` *Loading of MS library data*

Description

Loads, parses, verifies and curates MS library data, *e.g.* obtained from MassBank.

Usage

```
loadMSLibrary(file, algorithm, ...)
```

Arguments

`file`	A `character` string that specifies the the file path to the library.
`algorithm`	A character string describing the algorithm that should be used: `"msp"`, `"json"`
`...`	Any parameters to be passed to the selected MS library loading algorithm.

Details

`loadMSLibrary` is a generic function that will loads MS library data by one of the supported algorithms. The actual functionality is provided by algorithm specific functions such as `loadMSLibraryMSP` and `loadMSLibraryMoNAJSON`. While these functions may be called directly, `loadMSLibrary` provides a generic interface and is therefore usually preferred.

Value

A `MSLibrary` object containing the loaded library data.

See Also

The `MSLibrary` output class and its methods and the algorithm specific functions: `loadMSLibraryMSP`, `loadMSLibraryMoNAJSON`

loadMSLibraryMoNAJSON *Load MS library data from MassBank of North America (*MONA*)*

Description

This function loads, verifies and curates MS library data from MoNA '.json' files.

Usage

```
loadMSLibraryMoNAJSON(
  file,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE,
  potAdducts = TRUE,
  potAdductsLib = TRUE,
  absMzDev = 0.002,
  calcSPLASH = TRUE
)
```

Arguments

file	A character string that specifies the file path to the JSON library.
prefCalcChemProps	If TRUE then calculated chemical properties such as the formula and INCHIKEY are preferred over what is already present in the MS library. For efficiency reasons it is recommended to set this to TRUE. See the Validating and calculating chemical properties section for more details.
neutralChemProps	If TRUE then the neutral form of the molecule is considered to calculate SMILES, formulae etc. Enabling this may improve feature matching when considering common adducts (*e.g.* [M+H]+, [M-H]-). See the Validating and calculating chemical properties section for more details.
potAdducts, potAdductsLib	If and how missing adducts (Precursor_type data) are guessed, potAdducts should be either: • FALSE: do not perform adduct guessing. • TRUE: guesses adducts based on a common set of known adducts (currently based on GenFormAdducts and MetFragAdducts). If potAdductsLib is TRUE then also any adducts specified in the library are used. • A list with adduct objects or character vector that can be converted with as.adduct. Only the specified adducts will be used for guessing missing values.
absMzDev	The maximum absolute *m/z* deviation when guessing missing adducts.
calcSPLASH	If set to TRUE then missing SPLASH values will be calculated (see below).

Details

This function uses an efficient C++ JSON loader to load MS library data. This function is called when calling loadMSLibrary with algorithm="json".

This function uses C++ with **Rcpp** and **rapidjsonr** to efficiently load and parse JSON files from MoNA. An advantage compared to loadMSLibraryMSP is that this function supports loading spectral annotations.

The record field names are converted to those used in '.msp' files.

Value

The loaded data is returned in an MSLibrary object.

Automatic curation of library data

Several strategies are applied to automatically verify and improve library data. This is important, since library records may have inconsistent or erroneous data, which makes them unsuitable in automated workflows such as compounds annotation with generateCompoundsLibrary.

The loaded library data is post-treated as follows:

- The DB# field is renamed to DB_ID to improve compatibility with R column names.
- Synonyms (Synon fields) are merged together, mainly to save memory usage.
- Inconsistently formatted NA data (*e.g.* "n/a", "N/A" or empty strings) are set to regular R NA values.
- The case of record field names are made consistent.
- The Formula and ExactMass fields are renamed to formula and neutralMass, respectively. This is for consistency with other data generated with **patRoon**.
- character field data is trimmed from leading/trailing whitespace.
- Mass data is verified to be properly numeric, and set to NA otherwise.
- The format of formulae data is made consistent: ionic species (with or without square brackets) or converted to a regular formula format.
- Chemical identifiers such as SMILES and formulae are verified and missing values are calculated if possible. See below for more details.
- Shortened data in the Ion_mode field (P/N) is converted to the long format (POSITIVE/NEGATIVE).
- Many different adduct flavors typically found as Precursor_type data are converted and normalized to the generic textual format used by **patRoon** (see as.adduct).
- If potAdducts!=FALSE then missing or invalid adduct data in Precursor_type is guessed based on the difference between the neutral and ionic mass. If multiple adducts explain the mass difference the result is NA.
- Missing ion *m/z* data (PrecursorMZ field) is calculated from adduct data, if possible.
- Missing SPLASH data is calculated with the **splashR** package if calcSPLASH=TRUE.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formula in the MS library are automatically validated and calculated if missing/invalid.

The internal validation/calculation process performs the following steps:

- Validation of SMILES, INCHI, INCHIKEY and formula data (if present). Invalid entries will be set to NA.
- If `neutralChemProps=TRUE` then chemical data (SMILES, formulae etc.) is neutralized by (de-)protonation (using the `--neutralized` option of `OpenBabel`). An additional column `molNeutralized` is added to mark those molecules that were neutralized. Note that neutralization requires either SMILES or INCHI data to be available.
- The SMILES and INCHI data are used to calculate missing or invalid SMILES, INCHI, INCHIKEY and formula data. If `prefCalcChemProps=TRUE` then existing INCHIKEY and formula data is overwritten by calculated values whenever possible.
- The chemical formulae which were *not* calculated are verified and normalized. This process may be time consuming, and is potentially largely avoided by setting `prefCalcChemProps=TRUE`.
- Neutral masses are calculated for missing values (`prefCalcChemProps=FALSE`) or whenever possible (`prefCalcChemProps=TRUE`).

Note that calculation of formulae for molecules that are isotopically labelled is currently only supported for deuterium (2H) elements.

This functionality relies heavily on OpenBabel, please make sure it is installed.

Source

Guessing adducts from neutral/ionic mass differences was inspired from MetFrag.

References

Wohlgemuth G, Mehta SS, Mejia RF, Neumann S, Pedrosa D, Pluskal T, Schymanski EL, Willighagen EL, Wilson M, Wishart DS, Arita M, Dorrestein PC, Bandeira N, Wang M, Schulze T, Salek RM, Steinbeck C, Nainala VC, Mistrik R, Nishioka T, Fiehn O (2016). "SPLASH, a hashed identifier for mass spectra." *Nature Biotechnology*, **34**(11), 1099–1101. doi:10.1038/nbt.3689.

Ruttkies C, Schymanski EL, Wolf S, Hollender J, Neumann S (2016). "MetFrag relaunched: incorporating strategies beyond in silico fragmentation." *Journal of Cheminformatics*, **8**(1). doi:10.1186/s1332101601159.

Eddelbuettel D (2013). *Seamless R and C++ Integration with Rcpp*. Springer, New York. doi:10.1007/9781461468684, ISBN 978-1-4614-6867-7.

Eddelbuettel D, Balamuta J (2018). "Extending R with C++: A Brief Introduction to Rcpp." *The American Statistician*, **72**(1), 28-36. doi:10.1080/00031305.2017.1375990.

Eddelbuettel D, Francois R, Allaire J, Ushey K, Kou Q, Russell N, Ucar I, Bates D, Chambers J (2025). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.0, https://www.rcpp.org.

Eddelbuettel D, François R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

See Also

`loadMSLibrary` for more details and other algorithms.

The `MSLibrary` documentation for various methods to post-process the data and `generateCompoundsLibrary` for annotation of features with the library data.

`loadMSLibraryMSP` *Load MS library data from MSP files*

Description

This function loads, verifies and curates MS library data from MSP files.

Usage

```
loadMSLibraryMSP(
  file,
  parseComments = TRUE,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE,
  potAdducts = TRUE,
  potAdductsLib = TRUE,
  absMzDev = 0.002,
  calcSPLASH = TRUE
)
```

Arguments

`file`	A `character` string that specifies the file path to the MSP library.
`parseComments`	If `TRUE` then comments in the file are parsed to obtain additional fields, such as SMILES, `PubChemCID` and `Resolution`. Note that some records specify this data either in the comments or as a regular field, hence, to ensure that loaded data is most complete it is recommend to set `parseComments=TRUE`.
`prefCalcChemProps`	If `TRUE` then calculated chemical properties such as the formula and INCHIKEY are preferred over what is already present in the MS library. For efficiency reasons it is recommended to set this to `TRUE`. See the `Validating and calculating chemical properties` section for more details.

neutralChemProps
: If TRUE then the neutral form of the molecule is considered to calculate SMILES, formulae etc. Enabling this may improve feature matching when considering common adducts (*e.g.* [M+H]+, [M-H]-). See the Validating and calculating chemical properties section for more details.

potAdducts, potAdductsLib
: If and how missing adducts (Precursor_type data) are guessed, potAdducts should be either:
  - FALSE: do not perform adduct guessing.
  - TRUE: guesses adducts based on a common set of known adducts (currently based on GenFormAdducts and MetFragAdducts). If potAdductsLib is TRUE then also any adducts specified in the library are used.
  - A list with adduct objects or character vector that can be converted with as.adduct. Only the specified adducts will be used for guessing missing values.

absMzDev
: The maximum absolute *m/z* deviation when guessing missing adducts.

calcSPLASH
: If set to TRUE then missing SPLASH values will be calculated (see below).

Details

This function uses an efficient C++ MSP loader to load MS library data. This function is called when calling loadMSLibrary with algorithm="msp".

This function uses C++ with **Rcpp** to efficiently load and parse MSP files, and is mainly optimized for loading the '.msp' files from MassBank EU and MoNA. Files from other sources may also work, any feedback on this is welcome!

Value

The loaded data is returned in an MSLibrary object.

Automatic curation of library data

Several strategies are applied to automatically verify and improve library data. This is important, since library records may have inconsistent or erroneous data, which makes them unsuitable in automated workflows such as compounds annotation with generateCompoundsLibrary.

The loaded library data is post-treated as follows:

- The DB# field is renamed to DB_ID to improve compatibility with R column names.
- Synonyms (Synon fields) are merged together, mainly to save memory usage.
- Inconsistently formatted NA data (*e.g.* "n/a", "N/A" or empty strings) are set to regular R NA values.
- The case of record field names are made consistent.
- The Formula and ExactMass fields are renamed to formula and neutralMass, respectively. This is for consistency with other data generated with **patRoon**.
- character field data is trimmed from leading/trailing whitespace.
- Mass data is verified to be properly numeric, and set to NA otherwise.

- The format of formulae data is made consistent: ionic species (with or without square brackets) or converted to a regular formula format.
- Chemical identifiers such as SMILES and formulae are verified and missing values are calculated if possible. See below for more details.
- Shortened data in the `Ion_mode` field (P/N) is converted to the long format (`POSITIVE/NEGATIVE`).
- Many different adduct flavors typically found as `Precursor_type` data are converted and normalized to the generic textual format used by **patRoon** (see `as.adduct`).
- If `potAdducts!=FALSE` then missing or invalid adduct data in `Precursor_type` is guessed based on the difference between the neutral and ionic mass. If multiple adducts explain the mass difference the result is `NA`.
- Missing ion *m/z* data (`PrecursorMZ` field) is calculated from adduct data, if possible.
- Missing SPLASH data is calculated with the **splashR** package if `calcSPLASH=TRUE`.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formula in the MS library are automatically validated and calculated if missing/invalid.

The internal validation/calculation process performs the following steps:

- Validation of SMILES, INCHI, INCHIKEY and formula data (if present). Invalid entries will be set to `NA`.
- If `neutralChemProps=TRUE` then chemical data (SMILES, formulae etc.) is neutralized by (de-)protonation (using the `--neutralized` option of `OpenBabel`). An additional column `molNeutralized` is added to mark those molecules that were neutralized. Note that neutralization requires either SMILES or INCHI data to be available.
- The SMILES and INCHI data are used to calculate missing or invalid SMILES, INCHI, INCHIKEY and formula data. If `prefCalcChemProps=TRUE` then existing INCHIKEY and formula data is overwritten by calculated values whenever possible.
- The chemical formulae which were *not* calculated are verified and normalized. This process may be time consuming, and is potentially largely avoided by setting `prefCalcChemProps=TRUE`.
- Neutral masses are calculated for missing values (`prefCalcChemProps=FALSE`) or whenever possible (`prefCalcChemProps=TRUE`).

Note that calculation of formulae for molecules that are isotopically labelled is currently only supported for deuterium (2H) elements.

This functionality relies heavily on OpenBabel, please make sure it is installed.

Note

The mass spectrum parser currently only supports space separated entries (MSP formerly also allows other formats).

Source

Guessing adducts from neutral/ionic mass differences was inspired from MetFrag.

References

Wohlgemuth G, Mehta SS, Mejia RF, Neumann S, Pedrosa D, Pluskal T, Schymanski EL, Willighagen EL, Wilson M, Wishart DS, Arita M, Dorrestein PC, Bandeira N, Wang M, Schulze T, Salek RM, Steinbeck C, Nainala VC, Mistrik R, Nishioka T, Fiehn O (2016). "SPLASH, a hashed identifier for mass spectra." *Nature Biotechnology*, **34**(11), 1099–1101. doi:10.1038/nbt.3689.

Ruttkies C, Schymanski EL, Wolf S, Hollender J, Neumann S (2016). "MetFrag relaunched: incorporating strategies beyond in silico fragmentation." *Journal of Cheminformatics*, **8**(1). doi:10.1186/s1332101601159.

Eddelbuettel D (2013). *Seamless R and C++ Integration with Rcpp*. Springer, New York. doi:10.1007/9781461468684, ISBN 978-1-4614-6867-7.

Eddelbuettel D, Balamuta J (2018). "Extending R with C++: A Brief Introduction to Rcpp." *The American Statistician*, **72**(1), 28-36. doi:10.1080/00031305.2017.1375990.

Eddelbuettel D, Francois R, Allaire J, Ushey K, Kou Q, Russell N, Ucar I, Bates D, Chambers J (2025). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.0, https://www.rcpp.org.

Eddelbuettel D, François R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

See Also

`loadMSLibrary` for more details and other algorithms.

The `MSLibrary` documentation for various methods to post-process the data and `generateCompoundsLibrary` for annotation of features with the library data.

`makeHCluster` *Hierarchical clustering of compounds*

Description

Perform hierarchical clustering of structure candidates based on chemical similarity and obtain overall structural information based on the maximum common structure (MCS).

Usage

```
makeHCluster(obj, method = "complete", ...)

## S4 method for signature 'compounds'
makeHCluster(
  obj,
```

```
  method,
  fpType = "extended",
  fpSimMethod = "tanimoto",
  maxTreeHeight = 1,
  deepSplit = TRUE,
  minModuleSize = 1
)
```

Arguments

obj	The compounds object to be clustered.
method	The clustering method passed to hclust.
...	further arguments specified to methods.
fpType	The type of structural fingerprint that should be calculated. See the type argument of the get.fingerprint function of **rcdk**.
fpSimMethod	The method for calculating similarities (i.e. not dissimilarity!). See the method argument of the fp.sim.matrix function of the **fingerprint** package.
maxTreeHeight, deepSplit, minModuleSize	Arguments used by cutreeDynamicTree.

Details

Often many possible chemical structure candidates are found for each feature group when performing compound annotation. Therefore, it may be useful to obtain an overview of their general structural properties. One strategy is to perform hierarchical clustering based on their chemical (dis)similarity, for instance, using the Tanimoto score. The resulting clusters can then be characterized by evaluating their *maximum common substructure* (MCS).

makeHCluster performs hierarchical clustering of all structure candidates for each feature group within a compounds object. The resulting dendrograms are automatically cut using the cutreeDynamicTree function from the **dynamicTreeCut** package. The returned compoundsCluster object can then be used, for instance, for plotting dendrograms and MCS structures and manually re-cutting specific clusters.

Value

makeHCluster returns an compoundsCluster object.

Source

The methodology applied here has been largely derived from 'chemclust.R' from the **metfRag** package and the package vignette of **rcdk**.

References

Guha R (2007). "Chemical Informatics Functionality in R." *Journal of Statistical Software*, **18**(6).

See Also

compoundsCluster

makeSet *Initiate sets workflows*

Description

Initiate sets workflows from specified feature data.

Usage

```
makeSet(obj, ...)

## S4 method for signature 'features'
makeSet(obj, ..., adducts, labels = NULL)

## S4 method for signature 'featuresSet'
makeSet(obj, ...)

## S4 method for signature 'featureGroups'
makeSet(
  obj,
  ...,
  groupAlgo,
  groupArgs = NULL,
  verbose = TRUE,
  adducts = NULL,
  labels = NULL
)

## S4 method for signature 'featureGroupsSet'
makeSet(obj, ...)
```

Arguments

obj, ...	features or featureGroups objects that should be used for the sets workflow.
adducts	The adduct assignments to each set. Should either be a list with adduct objects or a character vector (*e.g.* "[M+H]+"). The order should follow that of the objects given to the obj and ... arguments. For the featureGroups method: if NULL then adduct annotations are used.
labels	The labels, or *set names*, for each set to be created. The order should follow that of the objects given to the obj and ... arguments. If NULL, then labels are automatically generated from the polarity of the specified adducts argument (*e.g.* "positive", "negative").
groupAlgo	groupAlgo The name of the feature grouping algorithm. See the algorithm argument of groupFeatures for details.
groupArgs	A list with arguments directly passed to groupFeatures (can be named). Example: groupArgs=list(maxAlignMZ=0.002).
verbose	If set to FALSE then no text output is shown.

Details

The makeSet method function is used to initiate a sets workflow. The features from input objects are combined and then *neutralized* by replacing their *m/z* values by neutral monoisotopic masses. After neutralization features measured with *e.g.* different ionization polarities can be grouped since their neutral mass will be the same.

The analysis information for this object is updated with all analyses, and a set column is added to designate the set of each analysis. Note that currently, all analyses names **must** be unique across different sets.

makeSet supports two types of input:

1. features objects: makeSet combines the input objects into a featuresSet object, which is then grouped in the 'usual way' with groupFeatures.
2. featureGroups objects: In this case the features from the input objects are first neutralized and feature groups between sets are then combined with groupFeatures.

The advantage of the featureGroups method is that it preserves any adduct annotations already present (*e.g.* as set by selectIons or adducts<-). Furthermore, this approach allows more advanced workflows where the input featureGroups are first pre-treated with *e.g.* filter before the sets object is made. On the other hand, the features method is easier, as it doesn't require intermediate feature grouping steps and is often sufficient since adduct annotations can be made afterwards with selectIons/adducts<- and most filter operations do not need to be done per individual set.

The adduct information used for feature neutralization is specified through the adducts argument. Alternatively, when the featureGroups method of makeSet is used, then the adduct annotations already present in the input objects can also by used by setting adducts=NULL. The adduct information is also used to add adduct annotations to the output of makeSet.

Value

Either a featuresSet object (features method) or featureGroupsSet object (featureGroups method).

Note

Initiating a sets workflow recursively, *i.e.* with featuresSet or featureGroupsSet objects as input, is currently not supported.

---

newProject *Easily create new* **patRoon** *projects*

Description

The newProject function is used to quickly generate a processing R script. This tool allows the user to quickly select the targeted analyses, workflow steps and configuring some of their common parameters. This function requires to be run within a RStudio session. The resulting script is either added to the current open file or to a new file. The analysis information will be written to a '.csv' file so that it can easily be modified afterwards.

Usage

```
newProject(destPath = NULL)
```

Arguments

destPath	Set destination path value to this value (useful for debugging). Set to NULL for a default value.

---

optimizedParameters *Class containing optimization results.*

Description

Objects from this class contain optimization results resulting from design of experiment (DoE).

Usage

```
optimizedParameters(object, paramSet = NULL, DoEIteration = NULL)

optimizedObject(object, paramSet = NULL)

scores(object, paramSet = NULL, DoEIteration = NULL)

experimentInfo(object, paramSet, DoEIteration)

## S4 method for signature 'optimizationResult'
algorithm(obj)

## S4 method for signature 'optimizationResult'
length(x)

## S4 method for signature 'optimizationResult'
lengths(x, use.names = FALSE)

## S4 method for signature 'optimizationResult'
show(object)

## S4 method for signature 'optimizationResult,missing'
plot(
  x,
  paramSet,
  DoEIteration,
  paramsToPlot = NULL,
  maxCols = NULL,
  type = "contour",
  image = TRUE,
  contours = "colors",
```

```
  ...
)

## S4 method for signature 'optimizationResult'
optimizedParameters(object, paramSet = NULL, DoEIteration = NULL)

## S4 method for signature 'optimizationResult'
optimizedObject(object, paramSet = NULL)

## S4 method for signature 'optimizationResult'
scores(object, paramSet = NULL, DoEIteration = NULL)

## S4 method for signature 'optimizationResult'
experimentInfo(object, paramSet, DoEIteration)
```

Arguments

paramSet	Numeric index of the parameter set (*i.e.* the first parameter set gets index '1'). For some methods optional: if NULL the best will be selected.
DoEIteration	Numeric index specifying the DoE iteration within the specified paramSet. For some methods optional: if NULL the best will be selected.
obj, x, object	An optimizationResult object.
use.names	Ignored.
paramsToPlot	Which parameters relations should be plot. If NULL all will be plot. Alternatively, a list containing one or more character vectors specifying each two parameters that should be plotted. Finally, if only one pair should be plotted, can be a character vector specifying both parameters.
maxCols	Multiple parameter pairs are plotted in a grid. The maximum number of columns can be set with this argument. Set to NULL for no limit.
type	The type of plots to be generated: "contour", "image" or "persp". The equally named functions will be called for plotting.
image	Passed to contour (if type="contour").
contours	Passed to persp (if type="persp").
...	Further arguments passed to contour, image or persp (depending on type).

Details

Objects from this class are returned by optimizeFeatureFinding and optimizeFeatureGrouping.

Methods (by generic)

- algorithm(optimizationResult): Returns the algorithm that was used for finding features.
- length(optimizationResult): Obtain total number of experimental design iterations performed.
- lengths(optimizationResult): Obtain number of experimental design iterations performed for each parameter set.

- `show(optimizationResult)`: Shows summary information for this object.
- `plot(x = optimizationResult, y = missing)`: Generates response plots for all or a selected set of parameters.
- `optimizedParameters(optimizationResult)`: Returns parameter set yielding optimal results. The `paramSet` and `DoEIteration` arguments can be `NULL`.
- `optimizedObject(optimizationResult)`: Returns the object (*i.e.* a `features` or `featureGroups` object) that was generated with optimized parameters. The `paramSet` argument can be `NULL`.
- `scores(optimizationResult)`: Returns optimization scores. The `paramSet` and `DoEIteration` arguments can be `NULL`.
- `experimentInfo(optimizationResult)`: Returns a `list` with optimization information from an DoE iteration.

Slots

`algorithm` A character specifying the algorithm that was optimized.

`paramSets` A `list` with detailed results from each parameter set that was tested.

`bestParamSet` Numeric index of the parameter set yielding the best response.

Examples

```
## Not run:
# ftOpt is an optimization object.

# plot contour of all parameter pairs from the first parameter set/iteration.
plot(ftOpt, paramSet = 1, DoEIteration = 1)

# as above, but only plot two parameter pairs
plot(ftOpt, paramSet = 1, DoEIteration = 1,
     paramsToPlot = list(c("mzPPM", "chromFWHM"), c("chromFWHM", "chromSNR")))

# plot 3d perspective plots
plot(ftOpt, paramSet = 1, DoEIteration = 1, type = "persp")

## End(Not run)
```

---

parents *Base transformation products (TP) class*

Description

Holds information for all TPs for a set of parents.

Usage

```
parents(TPs)

products(TPs)

## S4 method for signature 'transformationProducts'
parents(TPs)

## S4 method for signature 'transformationProducts'
products(TPs)

## S4 method for signature 'transformationProducts'
length(x)

## S4 method for signature 'transformationProducts'
names(x)

## S4 method for signature 'transformationProducts'
show(object)

## S4 method for signature 'transformationProducts,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'transformationProducts,ANY,missing'
x[[i, j]]

## S4 method for signature 'transformationProducts'
x$name

## S4 method for signature 'transformationProducts'
as.data.table(x)

## S4 method for signature 'transformationProducts'
convertToSuspects(obj, includeParents = FALSE)

## S4 method for signature 'transformationProducts'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'transformationProducts'
filter(obj, properties = NULL, verbose = TRUE, negate = FALSE)
```

Arguments

TPs, x, obj, object	transformationProducts object to be accessed
i, j	For [/[[: A numeric or character value which is used to select parents by their index or name, respectively (for the order/names see names()).

	For `[`: Can also be logical to perform logical selection (similar to regular vectors). If missing all parents are selected. For `[[`: should be a scalar value. For `delete`: The data to remove from. `i` are the parents as numeric index, logical or character, `j` the transformation products as numeric index (row) or name of the TP. If either is `NULL` then data for all is removed. `j` may also be a function: it will be called for each parent, with the TP info table (a `data.table`), the parent name and any other arguments passed as `...` to `delete`. The return value of this function specifies the TP indices (rows) (specified as an `integer` or `logical` vector) or names to be removed.
`...`	For `delete`: passed to the function specified as `j`. Otherwise ignored.
`drop`	ignored.
`name`	The parent name (partially matched).
`includeParents`	If `TRUE` then parents are also included in the returned suspect list.
`properties`	A named `list` with properties to be filtered. Each item in the `list` should be named with the name of the property, and should be a vector with allowed values. To obtain the possible properties, run *e.g.* `names(TPs)[[1]]`. Example: `properties=list(likelihood=c("LIKELY","PROBABLE"))`. Set to `NULL` to ignore.
`verbose`	If set to `FALSE` then no text output is shown.
`negate`	If `TRUE` then filters are performed in opposite manner.

Details

This class holds all generated data for transformation products for a set of parents. The class is `virtual` and derived objects are created by TP generators.

The TP data in objects from this class include a `retDir` column. These are `numeric` values that hint what the the chromatographic retention order of a TP might be compared to its parent: a value of '`-1`' means it will elute earlier, '`1`' it will elute later and '`0`' that there is no significant difference or the direction is unknown. These values are based on a typical reversed phase separation. When structural information is available (*e.g.* when `generateTPsBioTransformer` or `generateTPsLibrary` was used to generate the data), the `retDir` values are based on calculated `log P` values of the parent and its TPs.

Value

`delete` returns the object for which the specified data was removed.

`filter` returns a filtered `transformationProducts` object.

Methods (by generic)

- `parents(transformationProducts)`: Accessor method for the `parents` slot of a `transformationProducts` class.
- `products(transformationProducts)`: Accessor method for the `products` slot.

- `length(transformationProducts)`: Obtain total number of transformation products.
- `names(transformationProducts)`: Obtain the names of all parents in this object.
- `show(transformationProducts)`: Show summary information for this object.
- `x[i`: Subset on parents.
- `x[[i`: Extracts a table with TPs for a parent.
- `$`: Extracts a table with TPs for a parent.
- `as.data.table(transformationProducts)`: Returns all TP data in a table.
- `convertToSuspects(transformationProducts)`: Converts this object to a suspect list that can be used as input for `screenSuspects`.
- `delete(transformationProducts)`: Completely deletes specified transformation product data.
- `filter(transformationProducts)`: Performs rule-based filtering. Useful to simplify and clean-up the data.

Slots

`parents` A `data.table` with metadata for all parents that have TPs in this object. Use the `parents` method for access.

`products` A `list` with `data.table` entries with TP information for each parent. Use the `products` method for access.

S4 class hierarchy

- `workflowStep`
  - `transformationProducts`
    - `transformationProductsStructure`
      - `transformationProductsStructureConsensus`
      - `transformationProductsCTS`
      - `transformationProductsBT`
      - `transformationProductsLibrary`
    - `transformationProductsFormula`
      - `transformationProductsLibraryFormula`
    - `transformationProductsLogic`

See Also

The derived `transformationProductsStructure` class for more methods and `generateTPs`

peakLists *Class containing MS Peak Lists*

Description

Contains all MS (and MS/MS where available) peak lists for a featureGroups object.

Usage

```
peakLists(obj, ...)

averagedPeakLists(obj, ...)

spectrumSimilarity(obj, ...)

## S4 method for signature 'MSPeakLists'
peakLists(obj)

## S4 method for signature 'MSPeakLists'
averagedPeakLists(obj)

## S4 method for signature 'MSPeakLists'
analyses(obj)

## S4 method for signature 'MSPeakLists'
groupNames(obj)

## S4 method for signature 'MSPeakLists'
length(x)

## S4 method for signature 'MSPeakLists'
show(object)

## S4 method for signature 'MSPeakLists,ANY,ANY,missing'
x[i, j, ..., reAverage = FALSE, drop = TRUE]

## S4 method for signature 'MSPeakLists,ANY,ANY'
x[[i, j]]

## S4 method for signature 'MSPeakLists'
x$name

## S4 method for signature 'MSPeakLists'
as.data.table(x, fGroups = NULL, averaged = TRUE)

## S4 method for signature 'MSPeakLists'
delete(obj, i = NULL, j = NULL, k = NULL, reAverage = FALSE, ...)
```

```
## S4 method for signature 'MSPeakLists'
filter(
  obj,
  absMSIntThr = NULL,
  absMSMSIntThr = NULL,
  relMSIntThr = NULL,
  relMSMSIntThr = NULL,
  topMSPeaks = NULL,
  topMSMSPeaks = NULL,
  minMSMSPeaks = NULL,
  isolatePrec = NULL,
  deIsotopeMS = FALSE,
  deIsotopeMSMS = FALSE,
  withMSMS = FALSE,
  annotatedBy = NULL,
  retainPrecursorMSMS = TRUE,
  reAverage = FALSE,
  negate = FALSE
)

## S4 method for signature 'MSPeakLists'
plotSpectrum(
  obj,
  groupName,
  analysis = NULL,
  MSLevel = 1,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'MSPeakLists'
spectrumSimilarity(
  obj,
  groupName1,
  groupName2 = NULL,
  analysis1 = NULL,
  analysis2 = NULL,
  MSLevel = 1,
  specSimParams = getDefSpecSimParams(),
  NAToZero = FALSE,
  drop = TRUE
)

## S4 method for signature 'MSPeakListsSet'
```

```
analysisInfo(obj)

## S4 method for signature 'MSPeakListsSet'
show(object)

## S4 method for signature 'MSPeakListsSet,ANY,ANY,missing'
x[i, j, ..., reAverage = FALSE, sets = NULL, drop = TRUE]

## S4 method for signature 'MSPeakListsSet'
as.data.table(x, fGroups = NULL, averaged = TRUE)

## S4 method for signature 'MSPeakListsSet'
delete(obj, i = NULL, j = NULL, k = NULL, reAverage = FALSE, ...)

## S4 method for signature 'MSPeakListsSet'
filter(
  obj,
  ...,
  annotatedBy = NULL,
  retainPrecursorMSMS = TRUE,
  reAverage = FALSE,
  negate = FALSE,
  sets = NULL
)

## S4 method for signature 'MSPeakListsSet'
plotSpectrum(
  obj,
  groupName,
  analysis = NULL,
  MSLevel = 1,
  title = NULL,
  specSimParams = getDefSpecSimParams(),
  xlim = NULL,
  ylim = NULL,
  perSet = TRUE,
  mirror = TRUE,
  ...
)

## S4 method for signature 'MSPeakListsSet'
spectrumSimilarity(
  obj,
  groupName1,
  groupName2 = NULL,
  analysis1 = NULL,
  analysis2 = NULL,
  MSLevel = 1,
```

```
  specSimParams = getDefSpecSimParams(),
  NAToZero = FALSE,
  drop = TRUE
)

## S4 method for signature 'MSPeakListsSet'
unset(obj, set)

getDefIsolatePrecParams(...)
```

Arguments

obj, x, object	The MSPeakLists object to access.
...	Further arguments passed to plot. For sets workflow methods: further arguments passed to the base MSPeakLists method.
i, j	For [/[[: A numeric or character value which is used to select analyses/feature groups by their index or name, respectively (for the order/names see analyses()/groupNames()). For [: Can also be logical to perform logical selection (similar to regular vectors). If missing all analyses/feature groups are selected. For [[: should be a scalar value. If j is not specified, i selects by feature groups instead. For delete: The data to remove from. i are the feature groups as numeric index, logical or character, j the MS peaks as numeric indices (rows). If either is NULL then data for all is removed. j may also be a function: it will be called for each feature group, with the peak list table (a data.table), feature group name, analysis (NULL if averaged peak list), type ("MS" or "MSMS") and any other arguments passed as ... to delete. The return value of this function specifies the peak list indices (rows) to be removed (specified as an integer or logical vector).
reAverage	Set to TRUE to regenerate group averaged MS peak lists. **NOTE** it is very important that any annotation data relying on MS peak lists (formulae/compounds) are regenerated afterwards! Otherwise it is likely that *e.g.* plotting methods will use wrong MS/MS data.
drop	If set to TRUE and if the comparison is made between two spectra then drop is used to reduce the matrix return value to a numeric vector.
name	The feature group name (partially matched).
fGroups	The featureGroups object that was used to generate this object. If not NULL it is used to add feature group information (retention and *m/z* values).
averaged	If TRUE then feature group averaged peak list data is used.
k	A vector with analyses (character with names or integer with indices) for which the data should be deleted. If k!=NULL then deletions will *not* occur on group averaged peak lists. Otherwise, if k=NULL then deletion occurs on *both* group averaged and analysis specific peak lists.

absMSIntThr, absMSMSIntThr, relMSIntThr, relMSMSIntThr
: Absolute/relative intensity threshold for MS or MS/MS peak lists. NULL for none.

topMSPeaks, topMSMSPeaks
: Only consider this amount of MS or MS/MS peaks with highest intensity. NULL to consider all.

minMSMSPeaks
: If the number of peaks in an MS/MS peak list (**excluding** the precursor peak) is lower than this it will be completely removed. Set to NULL to ignore.

isolatePrec
: If not NULL then value should be a list with parameters used for isolating the precursor and its isotopes in MS peak lists (see Isolating precursor data). Alternatively, TRUE to apply the filter with default settings (as given with getDefIsolatePrecParams).

deIsotopeMS, deIsotopeMSMS
: Remove any isotopic peaks in MS or MS/MS peak lists. This may improve data processing steps which do not assume the presence of isotopic peaks (e.g. MetFrag for MS/MS). Note that getMzRPeakLists does not (yet) support flagging of isotopes.

withMSMS
: If set to TRUE then only results will be retained for which MS/MS data is available. if negate=TRUE then only results *without* MS/MS data will be retained.

annotatedBy
: Either a formulas or compounds object, or a list with both. Any MS/MS peaks that are *not* annotated by any of the candidates in the specified objects are removed.

retainPrecursorMSMS
: If TRUE then precursor peaks will never be filtered out from MS/MS peak lists (note that precursors are never removed from MS peak lists). The negate argument does not affect this setting.

negate
: If TRUE then filters are applied in opposite manner.

groupName
: The name of the feature group for which a plot should be made. To compare spectra, two group names can be specified.

analysis
: The name of the analysis for which a plot should be made. If NULL then data from the feature group averaged peak list is used. When comparing spectra, either NULL or the analyses for both spectra should be specified.

MSLevel
: The MS level: '1' for regular MS, '2' for MSMS.

title
: The title of the plot. If NULL a title will be automatically made.

specSimParams
: A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.

xlim, ylim
: Sets the plot size limits used by plot. Set to NULL for automatic plot sizing.

groupName1, groupName2
: The names of the feature groups for which the comparison should be made. If both arguments are specified then a comparison is made with the spectra specified by groupName1 *vs* those specified by groupName2. The length of either can be '>1' to generate a comparison matrix. Alternatively, if groupName2 is NULL then all the spectra specified in groupName1 will be compared with eachother, *i.e.* resulting in a square similarity matrix.

`analysis1`, `analysis2`
: The name of the analysis (analyses) for the comparison. If `NULL` then data from the feature group averaged peak list is used. Otherwise, should be the same length as `groupName1`/`groupName2`.

`NAToZero`
: Set to `TRUE` to convert `NA` similarities (*i.e.* when no similarity could be calculated) to zero values.

`sets`
: (**sets workflow**) A `character` with name(s) of the sets to keep (or remove if `negate=TRUE`).

`perSet`, `mirror`
: (**sets workflow**) If `perSet=TRUE` then the set specific mass peaks are annotated separately. Furthermore, if `mirror=TRUE` (and there are two sets in the object) then a mirror plot is generated.

`set`
: (**sets workflow**) The name of the set.

Details

Objects for this class are returned by `generateMSPeakLists`.

The `getDefIsolatePrecParams` is used to create a parameter list for isolating the precursor and its isotopes (see `Isolating precursor data`).

Value

`peakLists` returns a nested list containing MS (and MS/MS where available) peak lists per feature group and per analysis. The format is: `[[analysis]][[featureGroupName]][[MSType]][[PeakLists]]` where `MSType` is either `"MS"` or `"MSMS"` and `PeakLists` a `data.table` containing all *m/z* values (`mz` column) and their intensities (`intensity` column). In addition, the peak list tables may contain a `cmp` column which contains an unique alphabetical identifier to which isotopic cluster (or "compound") a mass belongs (only supported by MS peak lists generated by Bruker tools at the moment).

`averagedPeakLists` returns a nested list of feature group averaged peak lists in a similar format as `peakLists`.

`delete` returns the object for which the specified data was removed.

Methods (by generic)

- `peakLists(MSPeakLists)`: Accessor method to obtain the MS peak lists.
- `averagedPeakLists(MSPeakLists)`: Accessor method to obtain the feature group averaged MS peak lists.
- `analyses(MSPeakLists)`: returns a `character` vector with the names of the analyses for which data is present in this object.
- `groupNames(MSPeakLists)`: returns a `character` vector with the names of the feature groups for which data is present in this object.
- `length(MSPeakLists)`: Obtain total number of *m/z* values.
- `show(MSPeakLists)`: Shows summary information for this object.
- `x[i`: Subset on analyses/feature groups.
- `x[[i`: Extract a list with MS and MS/MS (if available) peak lists. If the second argument (`j`) is not specified the averaged peak lists for the group specified by the first argument (`i`) will be returned.

- `$`: Extract group averaged MS peaklists for a feature group.
- `as.data.table(MSPeakLists)`: Returns all MS peak list data in a table.
- `delete(MSPeakLists)`: Completely deletes specified peaks from MS peak lists.
- `filter(MSPeakLists)`: provides post filtering of generated MS peak lists, which may further enhance quality of subsequent workflow steps (*e.g.* formulae calculation and compounds identification) and/or speed up these processes. The filters are applied to peak lists for each feature and feature group. The feature group peak lists are *not* re-averaged by default (see the `reAverage` argument). *not* filtered afterwards.
- `plotSpectrum(MSPeakLists)`: Plots a spectrum using MS or MS/MS peak lists for a given feature group. Two spectra can be compared when two feature groups are specified.
- `spectrumSimilarity(MSPeakLists)`: Calculates the spectral similarity between two or more spectra.

Slots

`peakLists` Contains a list of all MS (and MS/MS) peak lists. Use the `peakLists` method for access.

`metadata` Metadata for all spectra used to generate peak lists. Follows the format of the `peakLists` slot.

`averagedPeakLists` A `list` with averaged MS (and MS/MS) peak lists for each feature group.

`avgPeakListArgs` A `list` with arguments used to generate feature group averaged MS(/MS) peak lists.

`origFGNames` A `character` with the original input feature group names.

`analysisInfo` (**sets workflow**) Analysis information. Use the `analysisInfo` method for access.

Isolating precursor data

Formula calculation typically relies on evaluating the measured isotopic pattern from the precursor to score candidates. Some algorithms (currently only `GenForm`) penalize candidates if mass peaks are present in MS1 spectra that do not contribute to the isotopic pattern. Since these spectra are typically very 'noisy' due to background and co-eluting ions, an additional filtering step may be recommended prior to formula calculation. During this precursor isolation step all mass peaks are removed that are (1) not the precursor and (2) not likely to be an isotopologue of the precursor. To determine potential isotopic peaks the following parameters are used:

- `maxIsotopes` The maximum number of isotopes to consider. For instance, a value of '`5`' means that `M+0` (*i.e.* the monoisotopic peak) till `M+5` is considered. All mass peaks outside this range are removed.
- `mzDefectRange` A two-sized `vector` specifying the minimum (can be negative) and maximum *m/z* defect deviation compared to the precursor *m/z* defect. When chlorinated, brominated or other compounds with strong *m/z* defect in their isotopologues are to be considered a higher range may be desired. On the other hand, for natural compounds this range may be tightened. Note that the search range is propegated with increasing distance from the precursor, *e.g.* the search range is doubled for `M+2`, tripled for `M+3` etc.

- `intRange` A two-sized `vector` specifying the minimum and maximum relative intensity range compared to the precursor. For instance, `c(0.001, 2)` removes all peaks that have an intensity below 0.1% or above 200% of that of the precursor.
- `z` The `z` value (*i.e.* absolute charge) to be considerd. For instance, a value of `2` would look for `M+0.5`, `M+1` etc. Note that the `mzDefectRange` is adjusted accordingly (*e.g.* halved if `z=2`).
- `maxGap` The maximum number of missing adjacent isotopic peaks ('gaps'). If the (rounded) *m/z* difference to the previous peak exceeds this value then this and all next peaks will be removed. Similar to `z`, the maximum gap is automatically adjusted for `charge`.

These parameters should be in a `list` that is passed to the `isolatePrec` argument to `filter`. The default values can be obtained with the `getDefIsolatePrecParams` function:

`maxIsotopes=5`; `mzDefectRange=c(-0.01, 0.01)`; `intRange=c(0.001, 2)`; `z=1`; `maxGap=2`

S4 class hierarchy

- `workflowStep`
  - `MSPeakLists`
    - `MSPeakListsSet`
    - `MSPeakListsUnset`

Source

`spectrumSimilarity`: The principles of spectral binning and cosine similarity calculations were loosely was based on the code from `SpectrumSimilarity()` function of **OrgMassSpecR**.

Sets workflows

The `MSPeakListsSet` class is applicable for sets workflows. This class is derived from `MSPeakLists` and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class `workflowStepSet`.
- `unset` Converts the object data for a specified set into a 'non-set' object (`MSPeakListsUnset`), which allows it to be used in 'regular' workflows. Only the MS peaks that are present in the specified set are kept.
- `analysisInfo` Returns the analysis info for this object.

The following methods are changed or with new functionality:

- `filter` and the subset operator (`[`) Can be used to select data that is only present for selected sets. The `filter` method is applied for each set individually, and afterwards the results are combined again (see `generateMSPeakLists`). Note that this has important implications for *e.g.* relative intensity filters (`relMSIntThr`/`relMSMSIntThr`), `topMSPeaks`/`topMSMSPeaks` and `minMSMSPeaks`. Similarly, when the `annotatedBy` filter is applied, each set specific MS peak list is filtered by the annotation results from only that set.
- `plotSpectrum` Is able to highlight set specific mass peaks (`perSet` and `mirror` arguments).

- `spectrumSimilarity` First calculates similarities for each spectral pair per set (*e.g.* all positive mode spectra are compared and then all negative mode spectra are compared). This data is then combined into an overall similarity value. How this combination is performed depends on the `setCombineMethod` field of the `specSimParams` argument.

Author(s)

For `spectrumSimilarity`: major contributions by Bas van de Velde for spectral binning and similarity calculation.

`plotHeatMap` *Components based on clustered intensity profiles.*

Description

This class is derived from `componentsClust` and is used to store hierarchical clustering information from intensity profiles of feature groups.

Usage

```
plotHeatMap(obj, ...)

## S4 method for signature 'componentsIntClust'
plotHeatMap(
  obj,
  interactive = FALSE,
  col = NULL,
  margins = c(6, 2),
  cexCol = 1,
  ...
)

## S4 method for signature 'componentsIntClust'
plotInt(
  obj,
  index,
  pch = 20,
  type = "b",
  lty = 3,
  col = NULL,
  plotArgs = NULL,
  linesArgs = NULL
)
```

Arguments

obj	A componentsIntClust object.
...	Further options passed to heatmap.2 / heatmaply (plotHeatMap).
interactive	If TRUE an interactive heatmap will be drawn (with heatmaply).
col	The colour used for plotting. Set to NULL for automatic colours.
margins, cexCol	Passed to heatmap.2
index	Numeric component/cluster index or component name.
pch, type, lty	Passed to lines.
plotArgs, linesArgs	A list with further arguments passed to plot and lines, respectively.

Details

Objects from this class are generated by generateComponentsIntClust

Value

plotHeatMap returns the same as heatmap.2 or heatmaply.

Methods (by generic)

- plotHeatMap(componentsIntClust): draws a heatmap using the heatmap.2 or heatmaply function.
- plotInt(componentsIntClust): makes a plot for all (normalized) intensity profiles of the feature groups within a given cluster.

Slots

clusterm Numeric matrix with normalized feature group intensities that was used for clustering.

S4 class hierarchy

- componentsClust
  - componentsIntClust

Note

When the object is altered (*e.g.* by filtering or subsetting it), methods that need the original clustered data such as plotting methods do not work anymore and stop with an error.

See Also

componentsClust for other relevant methods and generateComponents

`pred-aggr-params` *Parameters to aggregate concentrations/toxicity values assigned to feature groups*

Description

Parameters that are used by method functions such `as.data.table` to aggregate predicted concentrations or toxicities.

Usage

```
getDefPredAggrParams(all = mean, ...)
```

Arguments

`all`	The default aggregation function for all types, *e.g.* `mean`.
`...`	optional named arguments that override defaults.

Details

Multiple concentration or toxicity values may be assigned to a single feature group. To ease the interpretation and data handling, several functions aggregate these values prior their use. Aggregation occurs by the following data:

- The candidate (*i.e.* suspect or annotation candidate). This is mainly relevant for sets workflows, where calculations among sets may yield different results for the same candidate.
- The prediction type, *e.g.* all values that were obtained from suspect or compound annotation data.
- The feature group.

The aggregation of all data first occurs by the same candidate/type/feature group, then the same type/feature group and finally for each feature group. This ensures that *e.g.* large numbers of data points for a prediction type do not bias results.

The `candidateFunc`, `typeFunc` and `groupFunc` parameters specify the function that should be used to aggregate data. Commonly, functions such `mean`, `min` or `max` can be used here. Note that the function does not need to handle `NA` values, as these are removed in advance.

The `preferType` parameters specifies the *preferred* prediction type. Any values from other prediction types will be ignored unless the preferred type is not available for a feature group. Valid values are `"suspect"` (the default), `"compound"` (results from compound annotation by SMILES), `"SIRIUS_FP"` (results from formula/compound annotation with `SIRIUS+CSI:FingerID`) or `"none"`.

These parameters should be stored inside a `list`. The `getDefPredAggrParams` function can be used to generate such parameter list with defaults.

pred-quant *Functionality to predict quantitative data*

Description

Functions to predict response factors and feature concentrations from SMILES and/or SIRIUS+CSI:FingerID fingerprints using the **MS2Quant** package.

Usage

```
## S4 method for signature 'featureGroups'
calculateConcs(fGroups, featureAnn, areas = FALSE)

## S4 method for signature 'featureGroupsSet'
calculateConcs(fGroups, featureAnn, areas = FALSE)

## S4 method for signature 'compounds'
predictRespFactors(
  obj,
  fGroups,
  calibrants,
  eluent,
  organicModifier,
  pHAq,
  concUnit = "ugL",
  calibConcUnit = concUnit,
  updateScore = FALSE,
  scoreWeight = 1,
  parallel = TRUE
)

## S4 method for signature 'featureGroupsScreening'
predictRespFactors(
  obj,
  calibrants,
  eluent,
  organicModifier,
  pHAq,
  concUnit = "ugL",
  calibConcUnit = concUnit
)

## S4 method for signature 'featureGroupsScreening'
calculateConcs(fGroups, featureAnn = NULL, areas = FALSE)

## S4 method for signature 'featureGroupsScreeningSet'
predictRespFactors(obj, calibrants, ...)
```

```
## S4 method for signature 'featureGroupsScreeningSet'
calculateConcs(fGroups, featureAnn = NULL, areas = FALSE)

## S4 method for signature 'compoundsSet'
predictRespFactors(obj, fGroups, calibrants, ...)

## S4 method for signature 'compoundsSIRIUS'
predictRespFactors(
  obj,
  fGroups,
  calibrants,
  eluent,
  organicModifier,
  pHAq,
  concUnit = "ugL",
  calibConcUnit = concUnit,
  type = "FP"
)

## S4 method for signature 'formulasSet'
predictRespFactors(obj, fGroups, calibrants, ...)

## S4 method for signature 'formulasSIRIUS'
predictRespFactors(
  obj,
  fGroups,
  calibrants,
  eluent,
  organicModifier,
  pHAq,
  concUnit = "ugL",
  calibConcUnit = concUnit
)

getQuantCalibFromScreening(fGroups, concs, areas = FALSE, average = FALSE)
```

Arguments

fGroups	For predictRespFactors methods for feature annotations: The featureGroups object for which the annotations were performed. For calculateConcs: The featureGroups object for which concentrations should be calculated. For getQuantCalibFromScreening: A feature groups object screened for the calibrants with screenSuspects.
featureAnn	A featureAnnotations object (*e.g.* formulasSIRIUS or compounds) which contains response factors. Optional if calculateConcs is called on suspect screening results (*i.e.* featureGroupsScreening method).

areas — Set to TRUE to use peak areas instead of peak heights. Note: for calculateConcs this should follow what is in the calibrants table.

obj — The workflow object for which predictions should be performed, *e.g.* feature groups with screening results (featureGroupsScreening) or compound annotations (compounds).

calibrants — A data.frame with calibrants, see the Calibration section below.

(**sets workflow**) Should be a list with the calibrants for each set.

eluent — A data.frame that describes the LC gradient program. Should have a column time with the retention time in seconds and a column B with the corresponding percentage of the organic modifier ('0-100').

organicModifier — The organic modifier of the mobile phase: either "MeOH" (methanol) or "MeCN" (acetonitrile).

pHAq — The PH of the aqueous part of the mobile phase.

concUnit — The concentration unit for calculated concentrations. Can be molar based ("nM", "uM", "mM", "M") or mass based ("ngL", "ugL", "mgL", "gL"). Furthermore, can be prefixed with "log " for logarithmic concentrations (*e.g.* "log mM").

calibConcUnit — The concentration unit used in the calibrants table. For possible values see the concUnit argument.

updateScore, scoreWeight — If updateScore=TRUE then the annotation score column is updated by adding normalized values of the response factor (weighted by 'scoreWeight'). Currently, this **only** makes sense for annotations performed with MetFrag!

parallel — If set to TRUE then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.

... — (**sets workflow**) Further arguments passed to the non-sets workflow method.

type — Which types of predictions should be performed: should be "FP" (SIRIUS+CSI:FingerID fingerprints), "SMILES" or "both". Only relevant for compoundsSIRIUS method.

concs — A data.frame with concentration data. See the Calibration section below.

average — Set to TRUE to average intensity values within replicate groups.

Details

The MS2Quant R package predicts concentrations from SMILES and/or MS/MS fingerprints obtained with SIRIUS+CSI:FingerID. The predictRespFactors method functions interface with this package to calculate response factors, which can then be used to calculate feature concentrations with the calculateConcs method function.

Value

predictRespFactors returns an object amended with response factors (RF_SMILES/LRF_SIRFP columns).

calculateConcs returns a featureGroups based object amended with concentrations for each feature group (accessed with the concentrations method).

Calibration

The **MS2Quant** package requires calibration to convert predicted ionization efficiencies to instrument/method specific response factors. The calibration data should be specified with the `calibrants` argument to `predictRespFactors`. This should be a `data.frame` with intensity observations at different concentrations for a set of calibrants. Each row specifies one intensity observation at one concentration. The table should have the following columns:

- `name` The name of the calibrant. Can be freely chosen.
- `SMILES` The SMILES of the calibrant.
- `rt` The retention time of the calibrant (in seconds).
- `intensity` The peak intensity (or area, see the `areas` argument) of the calibrant.
- `conc` The concentration of the calibrant (see the `calibConcUnit` argument for specifying the unit).

It is recommended to include multiple calibrants (*e.g.* '`>=10`') at multiple concentrations (*e.g.* '`>=5`'). The latter is achieved by adding multiple rows for the same calibrant (keeping the `name`/`SMILES`/`rt` columns constant). It is also possible to follow the column naming used by **MS2Quant** (however retention times should still be in seconds!). For more details and tips see https://github.com/kruvelab/MS2Quant.

The `getQuantCalibFromScreening` function can be used to automatically generate a calibrants table from a feature groups object with suspect screening results. Here, the idea is to perform a screening with `screenSuspects` with a suspect list that contain the calibrants, which is then used to construct the calibrant table. It is highly recommended to add retention times for the calibrants in the suspect list to ensure the calibrant is assigned to the correct feature. Furthermore, it is possible to simply add the calibrants to the 'regular' suspect list in case a suspect screening was already part of the workflow. The `getQuantCalibFromScreening` function still requires you to specify concentration data, which is achieved via the `concs` argument. This should be a `data.frame` with a column `name` corresponding to the calibrant name (*i.e.* same as used by `screenSuspects` above) and columns with concentration data. The latter columns specify the concentrations of a calibrant in different replicate groups (as defined in the analysis information). The concentration columns should be named after the corresponding replicate group. Only those replicate groups that should be used for calibration need to be included. Furthermore, `NA` values can be used if a replicate group should be ignored for a specific calibrant.

Predicting response factors

The response factors are predicted with the `predictRespFactors` generic functions, which accepts the following input:

- Suspect screening results. The SMILES data is used to predict response factors for suspect hits.
- Formula annotation data obtained with `"sirius"` algorithm (`generateFormulasSIRIUS`). The predictions are performed for each formula candidate using `SIRIUS+CSI:FingerID` fingerprints. For this reason, the `getFingerprint` argument must be set to `TRUE` when generating the formula data.
- Compound annotation data obtained with the `"sirius"` algorithm (`generateCompoundsSIRIUS`). The predictions are performed for each annotation candidate using its SMILES and/or `SIRIUS+CSI:FingerID` fingerprints. The predictions are performed on a per formula basis, hence, response factors for isomers will be equal.

- Compound annotation data obtained with algorithms other than `"sirius"`. The response factors are predicted from SMILES data.

When SMILES data is used then predictions of response factors are generally more accurate. However, calculations with `SIRIUS+CSI:FingerID` fingerprints are faster and only require the formula and MS/MS spectrum, *i.e.* not the full structure. Hence, calculations with SMILES are mostly useful in suspect screening workflows, or with high confidence compound annotation data, whereas MS/MS fingerprints are suitable with unknowns.

For annotation data the calculations are performed for *all* candidates. This can especially lead to long running calculations when SMILES data is used. Hence, it is **strongly** recommended to first prioritize the annotation results, *e.g.* with the `topMost` argument to the filter method.

When response factors are predicted from `SIRIUS+CSI:FingerID` fingerprints then only formula and MS/MS spectra are used, even if compound annotations are used for input. The major difference is that with formula annotation input *all* formula candidates for which a fingerprint could be generated are considered, whereas with compound annotations only candidate formulae are considered for which also a structure could be assigned. Hence, the formula annotation input could be more comprehensive, whereas predictions from structure annotations could lead to more representative results as only formulae are considered for which at least one structure could be assigned.

Assigning concentrations

The `calculateConcs` generic function is used to assign concentrations for each feature using the response factors discussed in the previous section. The function takes response factors from suspect screening results and/or feature annotation data. If multiple response factors were predicted for the same feature group, for instance when multiple annotation candidates or suspect hits for this feature group are present, then a concentrations is assigned for all response factors. These values can later be easily aggregated with *e.g.* the as.data.table function.

Note

The **rcdk** package and OpenBabel tool are used internally to calculate molecular weights. Please make sure that `OpenBabel` is installed.

MS2Quant currently *only* supports '`M+H`' and '`M+`' adducts when performing predictions with `SIRIUS:FingerID` fingerprints. Predictions for candidates with other adducts, including '`M-H]`', are skipped with a warning.

References

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

Guha R (2007). "Chemical Informatics Functionality in R." *Journal of Statistical Software*, **18**(6).

Sepman H, Malm L, Peets P, MacLeod M, Martin J, Breitholtz M, Kruve A (2023). "Bypassing the Identification: MS2Quant for Concentration Estimations of Chemicals Detected with Non-target LC-HRMS from MS2 Data." *Analytical Chemistry*, **95**(33), 12329–12338. doi:10.1021/acs.analchem.3c01744, `https://doi.org/10.1021/acs.analchem.3c01744`.

See Also

Toxicity prediction

---

pred-tox *Functionality to predict toxicities*

Description

Functions to predict toxicities from SMILES and/or SIRIUS+CSI:FingerID fingerprints using the **MS2Tox** package.

Usage

```
## S4 method for signature 'featureGroups'
calculateTox(fGroups, featureAnn)

## S4 method for signature 'featureGroupsSet'
calculateTox(fGroups, featureAnn)

## S4 method for signature 'compounds'
predictTox(
  obj,
  LC50Mode = "static",
  concUnit = "ugL",
  updateScore = FALSE,
  scoreWeight = 1,
  parallel = TRUE
)

## S4 method for signature 'featureGroupsScreening'
predictTox(obj, LC50Mode = "static", concUnit = "ugL")

## S4 method for signature 'featureGroupsScreening'
calculateTox(fGroups, featureAnn = NULL)

## S4 method for signature 'featureGroupsScreeningSet'
predictTox(obj, ...)

## S4 method for signature 'featureGroupsScreeningSet'
calculateTox(fGroups, featureAnn = NULL)

## S4 method for signature 'compoundsSet'
predictTox(obj, ...)

## S4 method for signature 'compoundsSIRIUS'
predictTox(obj, type = "FP", LC50Mode = "static", concUnit = "ugL")
```

```
## S4 method for signature 'formulasSet'
predictTox(obj, ...)

## S4 method for signature 'formulasSIRIUS'
predictTox(obj, LC50Mode = "static", concUnit = "ugL")
```

Arguments

fGroups	For predictTox methods for feature annotations: The featureGroups object for which the annotations were performed. For calculateTox: The featureGroups object for which toxicities should be assigned.
featureAnn	A featureAnnotations object (*e.g.* formulasSIRIUS or compounds) which contains toxicities. Optional if calculateTox is called on suspect screening results (*i.e.* featureGroupsScreening method).
obj	The workflow object for which predictions should be performed, *e.g.* feature groups with screening results (featureGroupsScreening) or compound annotations (compounds).
LC50Mode	The mode used for predictions: should be "static" or "flow".
concUnit	The concentration unit for calculated toxicities. Can be molar based ("nM", "uM", "mM", "M") or mass based ("ngL", "ugL", "mgL", "gL"). Furthermore, can be prefixed with "log " for logarithmic concentrations (*e.g.* "log mM").
updateScore, scoreWeight	If updateScore=TRUE then the annotation score column is updated by adding normalized values of the response factor (weighted by 'scoreWeight'). Currently, this **only** makes sense for annotations performed with MetFrag!
parallel	If set to TRUE then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.
...	(**sets workflow**) Further arguments passed to the non-sets workflow method.
type	Which types of predictions should be performed: should be "FP" (SIRIUS+CSI:FingerID fingerprints), "SMILES" or "both". Only relevant for compoundsSIRIUS method.

Details

The MS2Tox R package predicts toxicities from SMILES and/or MS/MS fingerprints obtained with SIRIUS+CSI:FingerID. The predictTox method functions interface with this package to predict toxicities, which can then be assigned to feature groups with the calculateTox method function.

Value

predictTox returns an object amended with LC 50 values (LC50_SMILES/LC50_SIRFP columns).

calculateTox returns a featureGroups based object amended with toxicity values for each feature group (accessed with the toxicities method).

Predicting toxicities

The toxicities are predicted with the `predictTox` generic functions, which accepts the following input:

- Suspect screening results. The SMILES data is used to predict toxicities for suspect hits.
- Formula annotation data obtained with `"sirius"` algorithm (`generateFormulasSIRIUS`). The predictions are performed for each formula candidate using `SIRIUS+CSI:FingerID` fingerprints. For this reason, the `getFingerprint` argument must be set to `TRUE` when generating the formula data.
- Compound annotation data obtained with the `"sirius"` algorithm (`generateCompoundsSIRIUS`). The predictions are performed for each annotation candidate using its SMILES and/or `SIRIUS+CSI:FingerID` fingerprints. The predictions are performed on a per formula basis, hence, toxicities for isomers will be equal.
- Compound annotation data obtained with algorithms other than `"sirius"`. The toxicities are predicted from SMILES data.

When SMILES data is used then predictions of toxicities are generally more accurate. However, calculations with `SIRIUS+CSI:FingerID` fingerprints are faster and only require the formula and MS/MS spectrum, *i.e.* not the full structure. Hence, calculations with SMILES are mostly useful in suspect screening workflows, or with high confidence compound annotation data, whereas MS/MS fingerprints are suitable with unknowns.

For annotation data the calculations are performed for *all* candidates. This can especially lead to long running calculations when SMILES data is used. Hence, it is **strongly** recommended to first prioritize the annotation results, *e.g.* with the `topMost` argument to the filter method.

When toxicities are predicted from `SIRIUS+CSI:FingerID` fingerprints then only formula and MS/MS spectra are used, even if compound annotations are used for input. The major difference is that with formula annotation input *all* formula candidates for which a fingerprint could be generated are considered, whereas with compound annotations only candidate formulae are considered for which also a structure could be assigned. Hence, the formula annotation input could be more comprehensive, whereas predictions from structure annotations could lead to more representative results as only formulae are considered for which at least one structure could be assigned.

Assigning toxicities

The `calculateTox` generic function is used to assign toxicities for each feature using the toxicities discussed in the previous section. The function takes toxicities from suspect screening results and/or feature annotation data. If multiple toxicities were predicted for the same feature group, for instance when multiple annotation candidates or suspect hits for this feature group are present, then a toxicities is assigned for all toxicities. These values can later be easily aggregated with *e.g.* the as.data.table function.

Note

The **rcdk** package and OpenBabel tool are used internally to calculate molecular weights. Please make sure that `OpenBabel` is installed.

References

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

Guha R (2007). "Chemical Informatics Functionality in R." *Journal of Statistical Software*, **18**(6).

Peets P, Wang W, MacLeod M, Breitholtz M, Martin JW, Kruve A (2022). "MS2Tox Machine Learning Tool for Predicting the Ecotoxicity of Unidentified Chemicals in Water by Nontarget LC-HRMS." *Environmental Science & Technology*, **56**(22), 15508-15517. doi:10.1021/acs.est.2c02536, PMID: 36269851, https://doi.org/10.1021/acs.est.2c02536.

See Also

Concentration prediction

`printPackageOpts` *Prints all the package options of* `patRoon` *and their currently set values.*

Description

Prints all the package options of `patRoon` and their currently set values.

Usage

```
printPackageOpts()
```

`records` *Class to store data from a loaded MS library*

Description

Stores the spectra and metadata from the records of an MS library.

Usage

```
records(obj)

spectra(obj)

## S4 method for signature 'MSLibrary'
records(obj)

## S4 method for signature 'MSLibrary'
spectra(obj)
```

```
## S4 method for signature 'MSLibrary'
length(x)

## S4 method for signature 'MSLibrary'
names(x)

## S4 method for signature 'MSLibrary'
show(object)

## S4 method for signature 'MSLibrary,ANY,missing,missing'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'MSLibrary,ANY,missing'
x[[i, j]]

## S4 method for signature 'MSLibrary'
x$name

## S4 method for signature 'MSLibrary'
as.data.table(x)

## S4 method for signature 'MSLibrary'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'MSLibrary'
filter(
  obj,
  properties = NULL,
  massRange = NULL,
  mzRangeSpec = NULL,
  relMinIntensity = NULL,
  topMost = NULL,
  onlyAnnotated = FALSE,
  negate = FALSE
)

## S4 method for signature 'MSLibrary'
convertToSuspects(
  obj,
  adduct,
  spectrumType = "MS2",
 avgSpecParams = getDefAvgPListParams(minIntensityPre = 0, minIntensityPost = 2, topMost
    = 10),
  collapse = TRUE,
  suspects = NULL,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE
)
```

```
## S4 method for signature 'MSLibrary'
export(obj, type = "msp", out)

## S4 method for signature 'MSLibrary,MSLibrary'
merge(x, y, ...)
```

Arguments

x, obj, object	MSLibrary object to be accessed.
i	For [/[[: A numeric or character value which is used to select records by their index or name, respectively (for the order/names see names()). For [: Can also be logical to perform logical selection (similar to regular vectors). If missing all records are selected. For [[: should be a scalar value.
...	Unused.
drop, j	ignored.
name	The record name (partially matched).
properties	A named list with properties to be filtered. Each item in the list should be named with the name of the property, and should be a vector with allowed values. To obtain the possible properties, run *e.g.* names(records). Example: properties=list(Instrument_type=c("LC-ESI-QTOF","LC-ESI-TOF")). Set to NULL to ignore.
massRange	Records with a neutral mass outside this range will be removed. Should be a two-sized numeric vector with the lower and upper mass range. Set to NULL to ignore.
mzRangeSpec	Similar to the massRange argument, but removes any peaks from recorded mass spectra outside the given *m/z* range.
relMinIntensity	The minimum relative intensity ('0-1') of a mass peak to be kept. Set to NULL to ignore.
topMost	Only keep topMost number of mass peaks for each spectrum. This filter is applied after others. Set to NULL to ignore.
onlyAnnotated	If TRUE then only recorded spectra that are formula annotated are kept.
negate	If TRUE then filters are performed in opposite manner.
adduct	An adduct object (or something that can be converted to it with as.adduct). Any records with a different adduct (Precursor_type) are not considered. Alternatively, adduct can be set to NULL to not filter out any records. However, in this case *no* MS/MS fragments will be added to the returned suspect list.
spectrumType	A character vector which limits library records to the given spectrum types (Spectrum_type field, *e.g.* "MS2"). Set to NULL to allow all spectrum types.
avgSpecParams	A list with parameters used for averaging spectra. See getDefAvgPListParams for more details.

collapse — Whether records with the same first-block INCHIKEY should be collapsed. See the Suspect conversion section for details.

suspects — If not NULL then this should be a suspect list (see screenSuspects) which will be amended with spectra data. See the Suspect conversion section for details.

prefCalcChemProps — If TRUE then calculated chemical properties such as the formula and INCHIKEY are preferred over what is already present in the input suspect list to convertToSuspects. For efficiency reasons it is recommended to set this to TRUE. See the Validating and calculating chem section for more details.

neutralChemProps — If TRUE then the neutral form of the molecule is considered to calculate SMILES, formulae etc. Enabling this may improve feature matching when considering common adducts (*e.g.* [M+H]+, [M-H]-). See the Validating and calculating chemical properties section for more details.

type — The export type. Currently just "msp".

out — The file path to the output library file.

y — The MSLibrary to be merged with x.

Details

This class is used by loadMSLibrary to store the loaded MS library data.

Value

delete returns the object for which the specified data was removed.

filter returns a filtered MSLibrary object.

convertToSuspects return a suspect list (data.table), which can be used with screenSuspects.

merge returns a merged MSLibrary object.

Methods (by generic)

- records(MSLibrary): Accessor method for the records slot of an MSLibrary class.
- spectra(MSLibrary): Accessor method for the spectra slot of an MSLibrary class.
- length(MSLibrary): Obtains the total number of records stored.
- names(MSLibrary): Obtains the names of the stored records (DB_ID field).
- show(MSLibrary): Shows summary information for this object.
- x[i: Subset on records.
- x[[i: Extracts a spectrum table for a record.
- $: Extracts a spectrum table for a record.
- as.data.table(MSLibrary): Converts all the data (spectra and metadata) to a single data.table.
- delete(MSLibrary): Completely deletes specified full records or spectra.
- filter(MSLibrary): Performs rule-based filtering of records and spectra. This may be especially to improve annotation with generateCompoundsLibrary.

- `convertToSuspects(MSLibrary)`: Converts the MS library data to a suspect list, which can be used with `screenSuspects`. See the `Suspect conversion` section for details.
- `export(MSLibrary)`: Exports the library data to a '`.msp`' file. The export is accelerated by an `C++` interface with **Rcpp**.
- `merge(x = MSLibrary, y = MSLibrary)`: Merges two `MSLibrary` objects (`x` and `y`). The records from `y` that are unique are added to `x`. Records that were already in `x` are simply ignored. The SPLASH values are used to test equality between records, hence, the `calcSPLASH` argument to `loadMSLibrary` should be `TRUE`.

Slots

`records` A `data.table` with metadata for all records. Use the `records` method for access.

`spectra` A `list` with all (annotated) spectra. Each spectrum is stored in a `data.table`. Use the `spectra` method for access.

S4 class hierarchy

- `workflowStep`
  - `MSLibrary`

Suspect conversion

The `convertToSuspects` method converts MS library data to a suspect list, which can be used with *e.g.* `screenSuspects`. Furthermore, this function can also amend existing suspect lists with spectral data.

Conversion occurs in either of the following three methods:

1. *Direct* (`collapse=FALSE` and `suspects=NULL`): each record is considered a suspect, and the resulting suspect list is generated directly by converting the records metadata. The `fragments_mz` column for each suspect is constructed from the mass peaks of the corresponding record.
2. *Collapse* (`collapse=TRUE` and `suspects=NULL`): All records with the same first-block INCHIKEY are first merged, and their spectra are averaged using the parameters from the `avgSpecParams` argument (see `getDefAvgPListParams`). The suspect list is based on the merged records, where the `fragments_mz` column is constructed from the averaged spectra. This is generally a good default, especially with large MS libraries.
3. *Amend* (`suspects` is not `NULL`): only those records are considered if their first-block INCHIKEY is present in the suspect list. The remaining records and their spectra are then collapsed as described for the *Collapse* method, and the `fragments_mz` column for each suspect is set from the averaged spectra. If a suspect is not present in the library, its `fragments_mz` value will be empty. Note that any existing `fragments_mz` data will be overwritten.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formula in the input suspect list to `convertToSuspects` are automatically validated and calculated if missing/invalid.

The internal validation/calculation process performs the following steps:

- Validation of SMILES, INCHI, INCHIKEY and formula data (if present). Invalid entries will be set to NA.
- If neutralChemProps=TRUE then chemical data (SMILES, formulae etc.) is neutralized by (de-)protonation (using the --neutralized option of OpenBabel). An additional column molNeutralized is added to mark those molecules that were neutralized. Note that neutralization requires either SMILES or INCHI data to be available.
- The SMILES and INCHI data are used to calculate missing or invalid SMILES, INCHI, INCHIKEY and formula data. If prefCalcChemProps=TRUE then existing INCHIKEY and formula data is overwritten by calculated values whenever possible.
- The chemical formulae which were *not* calculated are verified and normalized. This process may be time consuming, and is potentially largely avoided by setting prefCalcChemProps=TRUE.
- Neutral masses are calculated for missing values (prefCalcChemProps=FALSE) or whenever possible (prefCalcChemProps=TRUE).

Note that calculation of formulae for molecules that are isotopically labelled is currently only supported for deuterium (2H) elements.

This functionality relies heavily on OpenBabel, please make sure it is installed.

Note

export does not split any Synon data that was merged when the library was loaded.

References

Wohlgemuth G, Mehta SS, Mejia RF, Neumann S, Pedrosa D, Pluskal T, Schymanski EL, Willighagen EL, Wilson M, Wishart DS, Arita M, Dorrestein PC, Bandeira N, Wang M, Schulze T, Salek RM, Steinbeck C, Nainala VC, Mistrik R, Nishioka T, Fiehn O (2016). "SPLASH, a hashed identifier for mass spectra." *Nature Biotechnology*, **34**(11), 1099–1101. doi:10.1038/nbt.3689.

Eddelbuettel D (2013). *Seamless R and C++ Integration with Rcpp*. Springer, New York. doi:10.1007/9781461468684, ISBN 978-1-4614-6867-7.

Eddelbuettel D, Balamuta J (2018). "Extending R with C++: A Brief Introduction to Rcpp." *The American Statistician*, **72**(1), 28-36. doi:10.1080/00031305.2017.1375990.

Eddelbuettel D, Francois R, Allaire J, Ushey K, Kou Q, Russell N, Ucar I, Bates D, Chambers J (2025). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.0, https://www.rcpp.org.

Eddelbuettel D, François R (2011). "Rcpp: Seamless R and C++ Integration." *Journal of Statistical Software*, **40**(8), 1–18. doi:10.18637/jss.v040.i08.

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

See Also

loadMSLibrary

replicateGroupSubtract *Filtering of grouped features*

Description

Basic rule based filtering of feature groups.

Usage

```
replicateGroupSubtract(fGroups, rGroups, threshold = 0)

## S4 method for signature 'featureGroups'
filter(
  obj,
  absMinIntensity = NULL,
  relMinIntensity = NULL,
  preAbsMinIntensity = NULL,
  preRelMinIntensity = NULL,
  absMinAnalyses = NULL,
  relMinAnalyses = NULL,
  absMinReplicates = NULL,
  relMinReplicates = NULL,
  absMinFeatures = NULL,
  relMinFeatures = NULL,
  absMinReplicateAbundance = NULL,
  relMinReplicateAbundance = NULL,
  absMinConc = NULL,
  relMinConc = NULL,
  absMaxTox = NULL,
  relMaxTox = NULL,
  absMinConcTox = NULL,
  relMinConcTox = NULL,
  maxReplicateIntRSD = NULL,
  blankThreshold = NULL,
  retentionRange = NULL,
  mzRange = NULL,
  mzDefectRange = NULL,
  chromWidthRange = NULL,
  featQualityRange = NULL,
  groupQualityRange = NULL,
  rGroups = NULL,
  results = NULL,
  removeBlanks = FALSE,
  removeISTDs = FALSE,
  checkFeaturesSession = NULL,
  predAggrParams = getDefPredAggrParams(),
```

```
  removeNA = FALSE,
  negate = FALSE
)

## S4 method for signature 'featureGroupsSet'
filter(
  obj,
  ...,
  negate = FALSE,
  sets = NULL,
  absMinSets = NULL,
  relMinSets = NULL
)

## S4 method for signature 'featureGroups'
replicateGroupSubtract(fGroups, rGroups, threshold = 0)
```

Arguments

fGroups, obj	featureGroups object to which the filter is applied.
rGroups	A character vector of replicate groups that should be kept (filter) or subtracted from (replicateGroupSubtract).
threshold	Minimum relative threshold (compared to mean intensity of replicate group being subtracted) for a feature group to be *not* removed. When '0' a feature group is always removed when present in the given replicate groups.
absMinIntensity, relMinIntensity	Minimum absolute/relative intensity for features to be kept. The relative intensity is determined from the feature with highest intensity (of all features from all groups). Set to '0' or NULL to skip this step.
preAbsMinIntensity, preRelMinIntensity	As absMinIntensity/relMinIntensity, but applied *before* any other filters. This is typically used to speed-up subsequent filter steps. However, care must be taken that a sufficiently low value is chosen that is not expected to affect subsequent filtering steps. See below why this may be important.
absMinAnalyses, relMinAnalyses	Feature groups are only kept when they contain data for at least this (absolute or relative) amount of analyses. Set to NULL to ignore.
absMinReplicates, relMinReplicates	Feature groups are only kept when they contain data for at least this (absolute or relative) amount of replicates. Set to NULL to ignore.
absMinFeatures, relMinFeatures	Analyses are only kept when they contain at least this (absolute or relative) amount of features. Set to NULL to ignore.
absMinReplicateAbundance, relMinReplicateAbundance	Minimum absolute/relative abundance that a grouped feature should be present within a replicate group. If this minimum is not met all features within the replicate group are removed. Set to NULL to skip this step.

absMinConc, relMinConc
: The minimum absolute/relative predicted concentration (calculated by calculateConcs) assigned to a feature. The toxicities are first aggregated prior to filtering, as controlled by the predAggrParams argument. Also see the removeNA argument.

absMaxTox, relMaxTox
: The maximum absolute/relative predicted toxicity (LC50) (calculated by calculateTox) assigned to a feature group. The concentrations are first aggregated prior to filtering, as controlled by the predAggrParams argument. Also see the removeNA argument.

absMinConcTox, relMinConcTox
: Like absMinConc/relMinConc, but instead considers the ratio between feature concentrations and the toxicity of the feature group. For instance, absMinConcTox=0.1 means that the calculated concentration of a feature should be at least '10%' of its toxicity.

maxReplicateIntRSD
: Maximum relative standard deviation (RSD) of intensity values for features within a replicate group. If the RSD is above this value all features within the replicate group are removed. Set to NULL to ignore.

blankThreshold
: Feature groups that are also present in blank analyses (see analysis info) are filtered out unless their relative intensity is above this threshold. For instance, a value of '5' means that only features with an intensity five times higher than that of the blank are kept. The relative intensity values between blanks and non-blanks are determined from the mean of all non-zero blank intensities. Set to NULL to skip this step.

retentionRange, mzRange, mzDefectRange, chromWidthRange
: Range of retention time (in seconds), *m/z*, mass defect (defined as the decimal part of *m/z* values) or chromatographic peak width (in seconds), respectively. Features outside this range will be removed. Should be a numeric vector with length of two containing the min/max values. The maximum can be Inf to specify no maximum range. Set to NULL to skip this step.

featQualityRange
: Used to filter features by their peak qualities/scores (see calculatePeakQualities). Should be a named list with min/max ranges for each quality/score to be filtered (the featureQualityNames function can be used to obtain valid names). Example: featQualityRange=list(ModalityScore=c(0.3, Inf),SymmetryScore=c(0.5, Inf)). Set to NULL to ignore.

groupQualityRange
: Like featQualityRange, but filters on group specific or averaged qualities/scores.

results
: Only keep feature groups that have results in the object specified by results. Valid classes are featureAnnotations (*e.g.* formula/compound annotations) and components. Can also be a list with multiple objects: in this case a feature group is kept if it has a result in *any* of the objects. Set to NULL to ignore.

removeBlanks
: Set to TRUE to remove all analyses that belong to replicate groups that are specified as a blank in the analysis-information. This is useful to simplify the analyses in the specified featureGroups object after blank subtraction. When both blankThreshold and this argument are set, blank subtraction is performed prior to removing any analyses.

`removeISTDs` If `TRUE` then all feature groups marked as internal standard (IS) are removed. This requires IS assignments done by `normInts`, see its documentation for more details.

`checkFeaturesSession`
If set then features and/or feature groups are removed that were selected for removal (see check-GUI). The session files are typically generated with the `checkFeatures` and `predictCheckFeaturesSession` functions. The value of `checkFeaturesSession` should either by a path to the session file or `TRUE`, in which case the default session file name is used. If `negate=TRUE` then all non-selected features/feature groups are removed instead.

`predAggrParams` Parameters to aggregate calculated concentrations/toxicities (obtained with `calculateConcs`/`calculateT` prior to filtering data. See prediction aggregation parameters for more information.

`removeNA` Set to `TRUE` to remove `NA` values. Currently only applicable to the concentration and toxicity filters.

`negate` If set to `TRUE` then filtering operations are performed in opposite manner.

`...` For sets workflow methods: further arguments passed to the base `featureGroups` method.

`sets` (**sets workflow**) A `character` with name(s) of the sets to keep (or remove if `negate=TRUE`).

`absMinSets`, `relMinSets`
(**sets workflow**) Feature groups are only kept when they contain data for at least this (absolute or relative) amount of sets. Set to `NULL` to ignore.

Details

`filter` performs common rule based filtering of feature groups such as blank subtraction, minimum intensity and minimum replicate abundance. Removing of features occurs by zeroing their intensity values. Furthermore, feature groups that are left completely empty (*i.e.* all intensities are zero) will be automatically removed.

`replicateGroupSubtract` removes feature groups present in a given set of replicate groups (unless intensities are above a given threshold). The replicate groups that are subtracted will be removed.

Value

A filtered `featureGroups` object. Feature groups that are filtered away have their intensity set to zero. In case a feature group is not present in any of the analyses anymore it will be removed completely.

Sets workflows

The following methods are changed or with new functionality:

- `filter` has specific arguments to filter by (feature presence in) sets. See the argument descriptions.

Filter order

When multiple arguments are specified to `filter`, multiple filters are applied in sequence. Since some of these filters may affect each other, choosing their order correctly may be important for effective data filtering. For instance, when an intensity filter removes features from blank analyses, a subsequent blank filter may not adequately perform blank subtraction. Similarly, when intensity and blank filters are executed after the replicate abundance filter it may be necessary to ensure minimum replicate abundance again as the intensity and blank filters may have removed some features within a replicate group.

With this in mind, filters (if specified) occur in the following order:

1. Features/feature groups selected for removal by the session specified by `checkFeaturesSession`.
2. Pre-Intensity filters (*i.e.* `preAbsMinIntensity` and `preRelMinIntensity`).
3. Chromatography and mass filters (*i.e* `retentionRange`, `mzRange`, `mzDefectRange`, `chromWidthRange`, `featQualityRange` and `groupQualityRange`).
4. Replicate abundance filters (*i.e.* `absMinReplicateAbundance`, `relMinReplicateAbundance` and `maxReplicateIntRSD`).
5. Blank filter (*i.e.* blankThreshold).
6. Intensity filters (*i.e.* `absMinIntensity` and `relMinIntensity`).
7. Replicate abundance filters (2nd time, only if previous filters affected results).
8. General abundance filters (*i.e.* `absMinAnalyses`, `relMinAnalyses`, `absMinReplicates`, `relMinReplicates`, `absMinFeatures`, `relMinFeatures`), `absMinConc`, `relMinConc`, `absMaxTox` and `relMaxTox`.
9. Replicate group filter (*i.e.* `rGroups`), results filter (*i.e.* `results`) and blank analyses / internal standard removal (*i.e.* `removeBlanks=TRUE` / `removeISTDs=TRUE`).

If another filtering order is desired then `filter` should be called multiple times with only one filter argument at a time.

See Also

`featureGroups-class` and `groupFeatures`

`report` *Report workflow data*

Description

Functionality to report data produced by most workflow steps such as features, feature groups, formula and compound annotations, and TPs.

Usage

```
report(
  fGroups,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  compsCluster = NULL,
  components = NULL,
  TPs = NULL,
  settingsFile = system.file("report", "settings.yml", package = "patRoon"),
  path = NULL,
  EICParams = getDefEICParams(topMost = 1, topMostByRGroup = TRUE),
  specSimParams = getDefSpecSimParams(),
  clearPath = FALSE,
  openReport = TRUE,
  parallel = TRUE,
  overrideSettings = list()
)

## S4 method for signature 'featureGroups'
report(
  fGroups,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  compsCluster = NULL,
  components = NULL,
  TPs = NULL,
  settingsFile = system.file("report", "settings.yml", package = "patRoon"),
  path = NULL,
  EICParams = getDefEICParams(topMost = 1, topMostByRGroup = TRUE),
  specSimParams = getDefSpecSimParams(),
  clearPath = FALSE,
  openReport = TRUE,
  parallel = TRUE,
  overrideSettings = list()
)

genReportSettingsFile(out = "report.yml", baseFrom = NULL)
```

Arguments

fGroups	The featureGroups object that should be used for reporting data.
MSPeakLists, formulas, compounds, compsCluster, components, TPs	Further objects (MSPeakLists, formulas, compounds, compoundsCluster, components, transformationProducts) that should be reported. Specify NULL to skip reporting a particular object. Note that MSPeakLists must be set if either formulas or compounds is set.

settingsFile	The path to the report settings file used for report configuration (see Report settings).
path	The destination file path for files generated during reporting. Will be generated if needed. If path=NULL then the destination path is taken from the report settings (see below).
EICParams	A named list with parameters used for extracted ion chromatogram (EIC) creation. See the EIC parameters documentation for more details.
specSimParams	A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.
clearPath	If TRUE then the report destination path will be (recursively) removed prior to reporting.
openReport	If set to TRUE then the output report file will be opened with the system browser.
parallel	If set to TRUE then code is executed in parallel through the **futures** package. Please see the parallelization section in the handbook for more details.
overrideSettings	A list with settings that override those from the report settings file. Example: overrideSettings=list(compounds=list(topMost=25)).
out	The output file path.
baseFrom	An existing report file to which the report settings should be based from. This is primarily used to update old settings files: the output settings file will be based on the old settings and amended with any missing.

Details

The reporting functionality is typically used at the very end of the workflow. It is used to overview the data generated during the workflow, such as features, their annotations and TP screening results.

report reports all workflow data in an interactive HTML file. The reports include both tabular data (*e.g.* retention times, annotation properties, screening results) and varios plots (*e.g.* chromatograms, (annotated) mass spectra and many more). This function uses functionality from other R packages, such as **rmarkdown**, **flexdashboard**, **knitr** and **bslib**.

The genReportSettingsFile function generates a new template 'YAML' file to configure report settings (see the next section).

Report settings

The report generation can be customized with a variety of settings that are read from a 'YAML' file. This is especially useful if you want to change more advanced settings or want to add or remove the parts that are reported The report settings file is specified through the settingsFile argument. If not specified then default settings will be used. To ease creation of a new template settings file, the genReportSettingsFile function can be used.

The following settings are currently available:

- General
  - format: the report format. Currently this can only be "html".
  - path: the destination path (ignored if the path argument is specified).
  - keepUnusedPlots: the number of days that unused plot files are kept (see Plot file caching).

  - selfContained: If true then the output 'report.html' embeds all graphics and script dependencies. Otherwise these files are read from the report_files/ directory. Self-contained reports are easily shared, since only the 'report.html' needs to be copied. However, they may be slower to generate and render, especially when the report contains a lot of data.
  - noDate Set to true to omit the date from the report. Mainly used for internal purposes.
- summary: defines the plots on the summary page: chord, venn and/or upset.
- features
  - retMin: if true then retention times are reported in minutes.
  - chromatograms
    * large: inclusion of large chromatograms (used in feature group table and TP parent chromatogram view).
    * small: inclusion of small chromatograms (feature group table).
    * features: inclusion of chromatograms for individual features (features view). Set to all to also include plots for analyses in which a feature was not found (or removed afterwards).
    * intMax: Method to determine the maximum intensity plot range: eic or feature. Sets the intMax argument to plotChroms.
  - intensityPlots: inclusion of intensity trend plots.
- MSPeakLists
  - spectra: inclusion of MS and MS/MS spectra (not annotated).
- formulas
  - include: whether formula results are reported (formula view). If false then the input formulas object is still used to amend *e.g.* compound annotated spectra.
  - normalizeScores, exclNormScores: controls score normalization, sets the equally named arguments to *e.g.* plotScores.
  - topMost only report this number of top ranked candidates. This number can be lowered to speed-up report generation.
- compounds
  - normalizeScores, exclNormScores, topMost: same as formulas, see above.
- TPs
  - graphs: inclusion of TP hierarchy graphs (generated with plotGraph).
  - graphStructuresMax: maximum number of structures to plot in hierarchy graphs (sets structuresMax argument of plotGraph).
- internalStandards
  - graph: inclusion of internal standard network plot (plotGraph).

Plot file caching

When a new report is generated the plot files are stored inside the report_files sub-directory inside the destination path of the report. The plot files are kept so they can be reused to speed-up re-creation of reports (*e.g.* with different report settings). After the report is generated, any unused plot files are removed unless they were recently created (controlled by the keepUnusedPlots setting, see previous section). The clearPath argument can be used to completely remove any old files.

Note

No data will be reported for feature groups in any of the reported objects (`formulas`, `compounds` etc) which are *not* present in the input `featureGroups` object (`fGroups`).

The `topMost`, `topMostByRGroup` and `onlyPresent` EIC parameters may be ignored, *e.g.*, when generating overview plots.

References

Creating MetFrag landing page URLs based on code from MetFamily R package.

Xie Y (2014). "knitr: A Comprehensive Tool for Reproducible Research in R." In Stodden V, Leisch F, Peng RD (eds.), *Implementing Reproducible Computational Research*. Chapman and Hall/CRC. ISBN 978-1466561595.

Xie Y (2015). *Dynamic Documents with R and knitr*, 2nd edition. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1498716963, https://yihui.org/knitr/.

Xie Y (2025). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.50, https://yihui.org/knitr/.

`reportCSV` *Report feature group data (legacy interface)*

Description

Functionality to report data produced by most workflow steps such as features, feature groups, calculated chemical formulae and tentatively identified compounds. This is the legacy interface, for the updated interface see reporting.

Usage

```
reportCSV(
  fGroups,
  path = "report",
  reportFeatures = FALSE,
  formulas = NULL,
  formulasNormalizeScores = "max",
  formulasExclNormScores = NULL,
  compounds = NULL,
  compoundsNormalizeScores = "max",
  compoundsExclNormScores = c("score", "individualMoNAScore", "annoTypeCount",
    "annotHitCount", "libMatch"),
  compsCluster = NULL,
  components = NULL,
  retMin = TRUE,
  clearPath = FALSE
```

```
)

reportPDF(
  fGroups,
  path = "report",
  reportFGroups = TRUE,
  formulas = NULL,
  formulasTopMost = 5,
  formulasNormalizeScores = "max",
  formulasExclNormScores = NULL,
  reportFormulaSpectra = TRUE,
  compounds = NULL,
  compoundsNormalizeScores = "max",
  compoundsExclNormScores = c("score", "individualMoNAScore", "annoTypeCount",
    "annotHitCount", "libMatch"),
  compoundsOnlyUsedScorings = TRUE,
  compoundsTopMost = 5,
  compsCluster = NULL,
  components = NULL,
  MSPeakLists = NULL,
  retMin = TRUE,
  EICGrid = c(2, 1),
 EICParams = getDefEICParams(rRtWindow = 20, topMost = 1, topMostByRGroup = TRUE),
  clearPath = FALSE
)

reportHTML(
  fGroups,
  path = "report",
  reportPlots = c("chord", "venn", "upset", "eics", "formulas"),
  formulas = NULL,
  formulasTopMost = 5,
  formulasNormalizeScores = "max",
  formulasExclNormScores = NULL,
  compounds = NULL,
  compoundsNormalizeScores = "max",
  compoundsExclNormScores = c("score", "individualMoNAScore", "annoTypeCount",
    "annotHitCount", "libMatch"),
  compoundsOnlyUsedScorings = TRUE,
  compoundsTopMost = 5,
  compsCluster = NULL,
  includeMFWebLinks = "compounds",
  components = NULL,
  interactiveHeat = FALSE,
  MSPeakLists = NULL,
  specSimParams = getDefSpecSimParams(),
  TPs = NULL,
  retMin = TRUE,
```

```
  EICParams = getDefEICParams(rtWindow = 20, topMost = 1, topMostByRGroup = TRUE),
  TPGraphStructuresMax = 25,
  selfContained = TRUE,
  optimizePng = FALSE,
  clearPath = FALSE,
  openReport = TRUE,
  noDate = FALSE
)

## S4 method for signature 'featureGroups'
reportCSV(
  fGroups,
  path = "report",
  reportFeatures = FALSE,
  formulas = NULL,
  formulasNormalizeScores = "max",
  formulasExclNormScores = NULL,
  compounds = NULL,
  compoundsNormalizeScores = "max",
  compoundsExclNormScores = c("score", "individualMoNAScore", "annoTypeCount",
    "annotHitCount", "libMatch"),
  compsCluster = NULL,
  components = NULL,
  retMin = TRUE,
  clearPath = FALSE
)

## S4 method for signature 'featureGroups'
reportPDF(
  fGroups,
  path = "report",
  reportFGroups = TRUE,
  formulas = NULL,
  formulasTopMost = 5,
  formulasNormalizeScores = "max",
  formulasExclNormScores = NULL,
  reportFormulaSpectra = TRUE,
  compounds = NULL,
  compoundsNormalizeScores = "max",
  compoundsExclNormScores = c("score", "individualMoNAScore", "annoTypeCount",
    "annotHitCount", "libMatch"),
  compoundsOnlyUsedScorings = TRUE,
  compoundsTopMost = 5,
  compsCluster = NULL,
  components = NULL,
  MSPeakLists = NULL,
  retMin = TRUE,
  EICGrid = c(2, 1),
```

```
  EICParams = getDefEICParams(),
  clearPath = FALSE
)

## S4 method for signature 'featureGroups'
reportHTML(
  fGroups,
  path = "report",
  reportPlots = c("chord", "venn", "upset", "eics", "formulas"),
  formulas = NULL,
  formulasTopMost = 5,
  formulasNormalizeScores = "max",
  formulasExclNormScores = NULL,
  compounds = NULL,
  compoundsNormalizeScores = "max",
  compoundsExclNormScores = c("score", "individualMoNAScore", "annoTypeCount",
    "annotHitCount", "libMatch"),
  compoundsOnlyUsedScorings = TRUE,
  compoundsTopMost = 5,
  compsCluster = NULL,
  includeMFWebLinks = "compounds",
  components = NULL,
  interactiveHeat = FALSE,
  MSPeakLists = NULL,
  specSimParams = getDefSpecSimParams(),
  TPs = NULL,
  retMin = TRUE,
 EICParams = getDefEICParams(rtWindow = 20, topMost = 1, topMostByRGroup = TRUE),
  TPGraphStructuresMax = 25,
  selfContained = TRUE,
  optimizePng = FALSE,
  clearPath = FALSE,
  openReport = TRUE,
  noDate = FALSE
)
```

Arguments

fGroups	The featureGroups object that should be used for reporting data.
path	The destination file path for files generated during reporting. Will be generated if needed.
reportFeatures	If set to TRUE then for each analysis a '.csv' file will be generated with information about its detected features.
formulas, compounds, compsCluster, components	Further objects (formulas, compounds, compoundsCluster, components) that should be reported. Specify NULL to skip reporting a particular object.
compoundsNormalizeScores, formulasNormalizeScores	A character that specifies how normalization of annotation scorings occurs.

Either "none" (no normalization), "max" (normalize to max value) or "minmax" (perform min-max normalization). Note that normalization of negative scores (e.g. output by SIRIUS) is always performed as min-max. Furthermore, currently normalization for compounds takes the original min/max scoring values into account when candidates were generated. Thus, for compounds scoring, normalization is not affected when candidate results were removed after they were generated (*e.g.* by use of filter).

compoundsExclNormScores, formulasExclNormScores

A character vector specifying any compound scoring names that should *not* be normalized. Set to NULL to normalize all scorings. Note that whether any normalization occurs is set by the compoundsExclNormScores,formulasExclNormScores argument.

For compounds: By default score and individualMoNAScore are set to mimic the behavior of the MetFrag web interface.

retMin If TRUE then report retention times in minutes (otherwise seconds).

clearPath If TRUE then the destination path will be (recursively) removed prior to reporting.

reportFGroups If TRUE then feature group data will be reported.

formulasTopMost, compoundsTopMost

Only this amount of top ranked candidate formulae/compounds are reported. Lower values may significantly speed up reporting. Set to NULL to ignore.

reportFormulaSpectra

If TRUE then explained MS/MS spectra (if available) for candidate formulae will be reported. Specifying formulas and setting this argument to FALSE still allows further annotation of compound MS/MS spectra.

compoundsOnlyUsedScorings

If TRUE then only scorings are plotted that actually have been used to rank data (see the scoreTypes argument to generateCompoundsMetFrag for more details).

MSPeakLists A MSPeakLists object that is *mandatory* when spectra for formulae and/or compounds will be reported.

EICGrid An integer vector in the form c(columns, rows) that is used to determine the plotting grid when reporting EICs in PDF files.

EICParams A named list with parameters used for extracted ion chromatogram (EIC) creation. See the EIC parameters documentation for more details.

reportPlots A character vector specifying what should be plotted. Valid options are: "chord", "venn", "upset" (plot a chord, Venn and UpSet diagram, respectively), "eics" (plot EICs for individual feature groups) and "formulas" (plot annotated formula spectra). Set to "none" to plot none of these.

includeMFWebLinks

A character specifying to which feature groups a web link should be added in the annotation page to MetFragWeb. Options are: "compounds" (only to those with compounds results), "MSMS" (only to those with MSMS peak lists) or "none".

interactiveHeat

If TRUE an interactive heatmap HTML widget will be generated to display hierarchical clustering results. Set to FALSE for a 'regular' static plot.

specSimParams — A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.

TPs — A transformationProducts object that should used for plotting hierarchies. Ignored if TPs=NULL or components is not a componentsTPs object.

TPGraphStructuresMax — Maximum number of TP structures to plot in TP hierarchies, see the TPs argument. Sets the structuresMax argument to plotGraph.

selfContained — If TRUE the output will be a standalone HTML file which contains all graphics and script dependencies. When FALSE, the latter will be placed in an additional directory ('report_files') which should remain present when viewing the output file. Especially on Windows, a non-self contained output might be desirable when reporting large amounts of data to prevent pandoc from running out of memory.

optimizePng — If TRUE then pngquant is used to reduce the size of generated graphics. A significant reduction in disk space usage may be seen, however, at the cost additional processing time. Multiple pngquant processes will be executed in parallel, which can be configured with 'patRoon.MP.maxProcs' (parallelization will always happen with the "classic" method, see patRoon options).

openReport — If set to TRUE then the output report file will be opened with the system browser.

noDate — If TRUE then the current date is not added to the report. This is mainly used for testing and its main purpose is to guarantees equal report files when reportHTML() is called multiple times with equal arguments.

Details

These functions are usually called at the very end of the workflow. It is used to report various data on features and feature groups. In addition, these functions may be used for reporting formulae and/or compounds that were generated for the specified feature groups. Data can be reported in tabular form (*i.e.* '.csv' files) by reportCSV or graphically by reportPDF and reportHTML. The latter functions will plot for instance chromatograms and annotated mass spectra, which are useful to get a graphical overview of results.

All functions have a wide variety of arguments that influence the reporting process. Nevertheless, most parameters are optional and only required to be given for fine tuning. In addition, only those objects (*e.g.* formulae, compounds, clustering) that are desired to be reported need to be specified.

reportCSV generates tabular data (*i.e.* '.csv' files) for given data to be reported. This may also be useful to allow import by other tools for post processing.

reportPDF will report graphical data (*e.g.* chromatograms and mass spectra) within PDF files. Compared to reportHTML this function may be faster and yield smaller report files, however, its functionality is a bit more basic and generated data is more 'scattered' around.

reportHTML will report graphical data (*e.g.* chromatograms and mass spectra) and summary information in an easy browsable HTML file using rmarkdown, flexdashboard and knitr.

Parallelization

reportHTML uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and patRoon options for configuration options.

Currently, reportHTML only uses "classic" multiprocessing, regardless of the 'patRoon.MP.method' option.

Note

Any formulae and compounds for feature groups which are not present within fGroups (*i.e.* because it has been subset afterwards) will not be reported.

The topMost, topMostByRGroup and onlyPresent EIC parameters may be ignored, *e.g.*, when generating overview plots.

References

Creating MetFrag landing page URLs based on code from MetFamily R package.

Xie Y (2014). "knitr: A Comprehensive Tool for Reproducible Research in R." In Stodden V, Leisch F, Peng RD (eds.), *Implementing Reproducible Computational Research*. Chapman and Hall/CRC. ISBN 978-1466561595.

Xie Y (2015). *Dynamic Documents with R and knitr*, 2nd edition. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1498716963, https://yihui.org/knitr/.

Xie Y (2025). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.50, https://yihui.org/knitr/.

See Also

reporting

---

screenInfo *Class for suspect screened feature groups.*

Description

This class derives from featureGroups and adds suspect screening information.

Usage

```
screenInfo(obj)

annotateSuspects(
  fGroups,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  ...
)
```

```
## S4 method for signature 'featureGroupsScreening'
screenInfo(obj)

## S4 method for signature 'featureGroupsScreening'
show(object)

## S4 method for signature 'featureGroupsScreening,ANY,ANY,missing'
x[i, j, ..., rGroups, suspects = NULL, drop = TRUE]

## S4 method for signature 'featureGroupsScreening'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featureGroupsScreening'
as.data.table(x, ..., collapseSuspects = ",", onlyHits = FALSE)

## S4 method for signature 'featureGroupsScreening'
annotateSuspects(
  fGroups,
  MSPeakLists,
  formulas,
  compounds,
  absMzDev = 0.005,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  checkFragments = c("mz", "formula", "compound"),
  formulasNormalizeScores = "max",
  compoundsNormalizeScores = "max",
  IDFile = system.file("misc", "IDLevelRules.yml", package = "patRoon"),
  logPath = file.path("log", "ident")
)

## S4 method for signature 'featureGroupsScreening'
filter(
  obj,
  ...,
  onlyHits = NULL,
  selectHitsBy = NULL,
  selectBestFGroups = FALSE,
  maxLevel = NULL,
  maxFormRank = NULL,
  maxCompRank = NULL,
  minAnnSimForm = NULL,
  minAnnSimComp = NULL,
  minAnnSimBoth = NULL,
  absMinFragMatches = NULL,
  relMinFragMatches = NULL,
  minRF = NULL,
  maxLC50 = NULL,
  negate = FALSE
```

```
)

## S4 method for signature 'featureGroupsScreeningSet'
screenInfo(obj)

## S4 method for signature 'featureGroupsScreeningSet'
show(object)

## S4 method for signature 'featureGroupsScreeningSet,ANY,ANY,missing'
x[i, j, ..., rGroups, suspects = NULL, sets = NULL, drop = TRUE]

## S4 method for signature 'featureGroupsScreeningSet'
delete(obj, i = NULL, j = NULL, ...)

## S4 method for signature 'featureGroupsScreeningSet'
as.data.table(x, ..., collapseSuspects = ",", onlyHits = FALSE)

## S4 method for signature 'featureGroupsScreeningSet'
annotateSuspects(
  fGroups,
  MSPeakLists,
  formulas,
  compounds,
  absMzDev = 0.005,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  checkFragments = c("mz", "formula", "compound"),
  formulasNormalizeScores = "max",
  compoundsNormalizeScores = "max",
  IDFile = system.file("misc", "IDLevelRules.yml", package = "patRoon"),
  logPath = file.path("log", "ident")
)

## S4 method for signature 'featureGroupsScreeningSet'
filter(
  obj,
  ...,
  onlyHits = NULL,
  selectHitsBy = NULL,
  selectBestFGroups = FALSE,
  maxLevel = NULL,
  maxFormRank = NULL,
  maxCompRank = NULL,
  minAnnSimForm = NULL,
  minAnnSimComp = NULL,
  minAnnSimBoth = NULL,
  absMinFragMatches = NULL,
  relMinFragMatches = NULL,
  minRF = NULL,
```

```
  maxLC50 = NULL,
  negate = FALSE
)

## S4 method for signature 'featureGroupsScreeningSet'
unset(obj, set)
```

Arguments

obj, object, x, fGroups	The featureGroupsScreening object.
MSPeakLists, formulas, compounds	Annotation data (MSPeakLists, formulas and compounds) obtained for this featureGroupsScreening object. All arguments can be NULL to exclude it from the annotation.
...	Further arguments passed to the base method.
i, j, rGroups	Used for subsetting data analyses, feature groups and replicate groups, see featureGroups.
suspects	An optional character vector with suspect names. If specified, only featureGroups will be kept that are assigned to these suspects.
drop	Ignored.
collapseSuspects	If a character then any suspects that were matched to the same feature group are collapsed to a single row and suspect names are separated by the value of collapseSuspects. If NULL then no collapsing occurs, and each suspect match is reported on a single row. See the Suspect collapsing section below for additional details.
onlyHits	For as.data.table: if TRUE then only feature groups with suspect hits are reported. For filter • if negate=FALSE and onlyHits=TRUE then all feature groups without suspect hits will be removed. Otherwise nothing will be done. • if negate=TRUE then onlyHits=TRUE will select feature groups without suspect hits, onlyHits=FALSE will only retain feature groups with suspect matches and this filter is ignored if onlyHits=NULL.
absMzDev	Maximum absolute *m/z* deviation.
specSimParams	A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.
checkFragments	Which type(s) of MS/MS fragments from workflow data should be checked to evaluate the number of suspect fragment matches (*i.e.* from the fragments_mz/fragments_formula columns in the suspect list). Valid values are: "mz", "formula", "compounds". The former uses *m/z* values in the specified MSPeakLists object, whereas the others use the formulae that were annotated to MS/MS peaks in the given formulas or compounds objects. Multiple values are possible: in this case the maximum number of fragment matches will be reported.

`compoundsNormalizeScores`, `formulasNormalizeScores`
: A `character` that specifies how normalization of annotation scorings occurs. Either `"max"` (normalize to max value) or `"minmax"` (perform min-max normalization). Note that normalization of negative scores (e.g. output by `SIRIUS`) is always performed as min-max. Furthermore, currently normalization for `compounds` takes the original min/max scoring values into account when candidates were generated. Thus, for `compounds` scoring, normalization is not affected when candidate results were removed after they were generated (*e.g.* by use of `filter`).

`IDFile`
: A file path to a YAML file with rules used for estimation of identification levels. See the `Suspect annotation` section for more details. If not specified then a default rules file will be used.

`logPath`
: A directory path to store logging information. If `NULL` then logging is disabled.

`selectHitsBy`
: Should be `"intensity"` or `"level"`. For cases where the same suspect is matched to multiple feature groups, only the suspect to the feature group with highest mean intensity (`selectHitsBy="intensity"`) or best identification level (`selectHitsBy="level"`) is kept. In case of ties only the first hit is kept. Set to `NULL` to ignore this filter. If `negate=TRUE` then only those hits with lowest mean intensity/poorest identification level are kept.

`selectBestFGroups`
: If `TRUE` then for any cases where a single feature group is matched to several suspects only the suspect assigned to the feature group with best identification score is kept. In case of ties only the first is kept.

`maxLevel`, `maxFormRank`, `maxCompRank`, `minAnnSimForm`, `minAnnSimComp`, `minAnnSimBoth`
: Filter suspects by maximum identification level (*e.g.* `"3a"`), formula/compound rank or with minimum formula/compound/combined annotation similarity. Set to `NULL` to ignore.

`absMinFragMatches`, `relMinFragMatches`
: Only retain suspects with this minimum number MS/MS matches with the fragments specified in the suspect list (*i.e.* `fragments_mz`/`fragments_formula`). `relMinFragMatches` sets the minimum that is relative (`'0-1'`) to the maximum number of MS/MS fragments specified in the `fragments_*` columns of the suspect list. Set to `NULL` to ignore.

`minRF`
: Filter suspect hits by the given minimum predicted response factor (as calculated by `predictRespFactors`). Set to `NULL` to ignore.

`maxLC50`
: Filter suspect hits by the given maximum toxicity (LC50) (as calculated by `predictTox`). Set to `NULL` to ignore.

`negate`
: If set to `TRUE` then filtering operations are performed in opposite manner.

`sets`
: (**sets workflow**) A `character` with name(s) of the sets to keep (or remove if `negate=TRUE`).

`set`
: (**sets workflow**) The name of the set.

Value

`annotateSuspects` returns a `featureGroupsScreening` object, which is a `featureGroups` object amended with annotation data.

filter returns a filtered featureGroupsScreening object.

Methods (by generic)

- screenInfo(featureGroupsScreening): Returns a table with screening information (see screenInfo slot).
- show(featureGroupsScreening): Shows summary information for this object.
- x[i: Subset on analyses, feature groups and/or suspects.
- as.data.table(featureGroupsScreening): Obtain a summary table (a data.table) with retention, *m/z*, intensity and optionally other feature data. Furthermore, the output table will be merged with information from screenInfo, such as suspect names and other properties and annotation data.
- annotateSuspects(featureGroupsScreening): Incorporates annotation data obtained during the workflow to annotate suspects with matched known MS/MS fragments, formula/candidate ranks and automatic estimation of identification levels. See the Suspect annotation section for more details. The estimation of identification levels for each suspect is logged in the log/ident directory.
- filter(featureGroupsScreening): Performs rule based filtering. This method builds on the comprehensive filter functionality from the base filter,featureGroups-method. It adds several filters to select *e.g.* the best ranked suspects or those with a minimum estimated identification level. **NOTE**: most filters *only* affect suspect hits, not feature groups. Set onlyHits=TRUE to subsequently remove any feature groups that lost any suspect matches due to other filter steps.

Slots

screenInfo A (data.table) with results from suspect screening. This table will be amended with annotation data when annotateSuspects is run.

MS2QuantMeta Metadata from **MS2Quant** filled in by predictRespFactors.

Suspect annotation

The annotateSuspects method is used to annotate suspects after screenSuspects was used to collect suspect screening results and other workflow steps such as formula and compound annotation steps have been completed. The annotation results, which can be acquired with the as.data.table and screenInfo methods, amends the current screening data with the following columns:

- formRank,compRank The rank of the suspect within the formula/compound annotation results.
- annSimForm,annSimComp,annSimBoth A similarity measure between measured and annotated MS/MS peaks from annotation of formulae, compounds or both. The similarity is calculated as the spectral similarity between a peaklist with (a) all MS/MS peaks and (b) only annotated peaks. Thus, a value of one means that all MS/MS peaks were annotated. If both formula and compound annotations are available then annSimBoth is calculated after combining all the annotated peaks, otherwise annSimBoth equals the available value for annSimForm or annSimComp. The similarity calculation can be configured with the specSimParams argument to annotateSuspects. Note for annotation with generateCompoundsLibrary results: the method and default parameters for annSimComp calculation slightly differs to those from

the spectral similarity calculated with compound annotation (`libMatch` score), hence small differences in results are typically observed.

- `maxFrags` The maximum number of MS/MS fragments that can be matched for this suspect (based on the `fragments_*` columns from the suspect list).
- `maxFragMatches`,`maxFragMatchesRel` The absolute and relative amount of experimental MS/MS peaks that were matched from the fragments specified in the suspect list. The value for `maxFragMatchesRel` is relative to the value for `maxFrags`. The calculation of this column is influenced by the `checkFragments` argument to `annotateSuspects`.
- `estIDLevel` Provides an *estimation* of the identification level, roughly following that of (Schymanski et al. 2014). However, please note that this value is only an estimation, and manual interpretation is still necessary to assign final identification levels. The estimation is done through a set of rules, see the `Identification level rules` section below.

Note that only columns are present if sufficient data is available for their calculation.

Identification level rules

The estimation of identification levels is configured through a YAML file which specifies the rules for each level. The default file is shown below.

```
1:
    suspectFragments: 3
    retention: 12
2a:
    or:
        - individualMoNAScore:
            min: 0.9
            higherThanNext: .inf
        - libMatch:
            min: 0.9
            higherThanNext: .inf
    rank:
        max: 1
        type: compound
3a:
    or:
        - individualMoNAScore: 0.4
        - libMatch: 0.4
3b:
    suspectFragments: 3
3c:
    annMSMSSim:
        type: compound
        min: 0.7
4a:
    annMSMSSim:
        type: formula
        min: 0.7
```

```
    isoScore:
        min: 0.5
        higherThanNext: 0.2
    rank:
        max: 1
        type: formula
4b:
    isoScore:
        min: 0.9
        higherThanNext: 0.2
    rank:
        max: 1
        type: formula
5:
    all: yes
```

Most of the file should be self-explanatory. Some notes:

- Each rule is either a field of `suspectFragments` (minimum number of MS/MS fragments matched from suspect list), `retention` (maximum retention deviation from suspect list), `rank` (the maximum annotation rank from formula or compound annotations), `all` (this level is always matched) or any of the scorings available from the formula or compound annotations.
- In case any of the rules could be applied to either formula or compound annotations, the annotation type must be specified with the `type` field (`formula` or `compound`).
- Identification levels should start with a number and may optionally be followed by a alphabetic character. The lowest levels are checked first.
- If `relative=yes` then the relative scoring will be used for testing.
- For `suspectFragments`: if the number of fragments from the suspect list (`maxFrags` column) is less then the minimum rule value, the minimum is adjusted to the number of available fragments.
- The `or` and `and` keywords can be used to combine multiple conditions.

A template rules file can be generated with the `genIDLevelRulesFile` function, and this file can subsequently passed to `annotateSuspects`. The file format is highly flexible and (sub)levels can be added or removed if desired. Note that the default file is currently only suitable when annotation is performed with GenForm and MetFrag, for other algorithms it is crucial to modify the rules.

S4 class hierarchy

- `featureGroups`
  - `featureGroupsScreening`
    - `featureGroupsSetScreeningUnset`

Suspect collapsing

The `as.data.table` method fir `featureGroupsScreening` supports an additional format where each suspect hit is reported on a separate row (enabled by setting `collapseSuspects=NULL`). In

this format the suspect properties from the screenInfo method are merged with each suspect row. Alternatively, if *suspect collapsing* is enabled (the default) then the regular as.data.table format is used, and amended with the names of all suspects matched to a feature group (separated by the value of the collapseSuspects argument).

Suspect collapsing also influences how calculated feature concentrations/toxicities are reported (*i.e.* obtained with calculateConcs/calculateTox). If these values were directly predicted for suspects, *i.e.* by using predictRespFactors/predictTox on the feature groups object, *and* suspects are *not* collapsed, then the calculated concentration/toxicity reported for each suspect row is not aggregated and specific for that suspect (unless not available). Hence, this allows you to obtain specific concentration/toxicity values for each suspect/feature group pair.

Sets workflows

The featureGroupsScreeningSet class is applicable for sets workflows. This class is derived from featureGroupsScreening and therefore largely follows the same user interface.

The following methods are specifically defined for sets workflows:

- All the methods from base class workflowStepSet.
- unset Converts the object data for a specified set into a 'non-set' object (featureGroupsScreeningUnset), which allows it to be used in 'regular' workflows. Only the screening results present in the specified set are kept.

The following methods are changed or with new functionality:

- annotateSuspects Suspect annotation is performed per set. Thus, formula/compound ranks, estimated identification levels etc are calculated for each set. Subsequently, these results are merged in the final screenInfo. In addition, an overall formRank and compRank column is created based on the rankings of the suspect candidate in the set consensus data. Furthermore, an overall estIDLevel is generated that is based on the 'best' estimated identification level among the sets data (*i.e.* the lowest). In case there is a tie between sub-levels (*e.g.* '3a' and '3b'), then the sub-level is stripped (*e.g.* '3').
- filter All filters related to estimated identification levels and formula/compound rankings are applied to the overall set data (see above). All others are applied to set specific data: in this case candidates are only removed if none of the set data confirms to the filter.

This class derives also from featureGroupsSet. Please see its documentation for more relevant details with sets workflows.

Note that the formRank and compRank columns are *not* updated when the data is subset.

Note

filter removes suspect hits with NA values when any of the filters related to minimum or maximum values are applied (unless negate=TRUE).

Author(s)

Rick Helmus <<r.helmus@uva.nl>>, Emma Schymanski <<emma.schymanski@uni.lu>> (contributions to identification level rules), Bas van de Velde (contributions to spectral similarity calculation).

References

Schymanski EL, Jeon J, Gulde R, Fenner K, Ruff M, Singer HP, Hollender J (2014). "Identifying Small Molecules via High Resolution Mass Spectrometry: Communicating Confidence." *Environmental Science and Technology*, **48**(4), 2097–2098. doi:10.1021/es5002105.

Stein SE, Scott DR (1994). "Optimization and testing of mass spectral library search algorithms for compound identification." *Journal of the American Society for Mass Spectrometry*, **5**(9), 859–866. doi:10.1016/10440305(94)870098.

See Also

featureGroups

---

screenSuspects *Target and suspect screening*

Description

Utilities to screen for analytes with known or suspected identity.

Usage

```
screenSuspects(
  fGroups,
  suspects,
  rtWindow = 12,
  mzWindow = 0.005,
  adduct = NULL,
  skipInvalid = TRUE,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE,
  onlyHits = FALSE,
  ...
)

## S4 method for signature 'featureGroups'
screenSuspects(
  fGroups,
  suspects,
  rtWindow,
  mzWindow,
  adduct,
  skipInvalid,
  prefCalcChemProps,
  neutralChemProps,
  onlyHits
)
```

```
## S4 method for signature 'featureGroupsScreening'
screenSuspects(
  fGroups,
  suspects,
  rtWindow,
  mzWindow,
  adduct,
  skipInvalid,
  onlyHits,
  amend = FALSE
)

numericIDLevel(level)

genIDLevelRulesFile(out, inLevels = NULL, exLevels = NULL)

## S4 method for signature 'featureGroupsSet'
screenSuspects(
  fGroups,
  suspects,
  rtWindow,
  mzWindow,
  adduct,
  skipInvalid,
  prefCalcChemProps,
  neutralChemProps,
  onlyHits
)

## S4 method for signature 'featureGroupsScreeningSet'
screenSuspects(
  fGroups,
  suspects,
  rtWindow,
  mzWindow,
  adduct,
  skipInvalid,
  prefCalcChemProps,
  neutralChemProps,
  onlyHits,
  amend = FALSE
)
```

Arguments

fGroups	The featureGroups object that should be screened.
suspects	A data.frame with suspect information. See the Suspect list format section

below.

(**sets workflow**) Can also be a `list` with suspect lists to be used for each set (otherwise the same suspect lists is used for all sets). The `list` can be named with the sets names to mark which suspect list is to be used with which set (*e.g.* `suspects=list(positive=suspsPos, negative=suspsNeg)`).

`rtWindow`, `mzWindow`
: The retention time window (in seconds) and *m/z* window that will be used for matching a suspect (+/- feature data).

`adduct`
: An `adduct` object (or something that can be converted to it with `as.adduct`). Examples: `"[M-H]-"`, `"[M+Na]+"`. May be `NULL`, see `Suspect list format` and `Matching of suspect masses` sections below.

`skipInvalid`
: If set to `TRUE` then suspects with invalid data (*e.g.* missing names or other missing data) will be ignored with a warning. Similarly, any suspects for which mass calculation failed (when no `mz` column is present in the suspect list), for instance, due to invalid `SMILES`, will be ignored with a warning.

`prefCalcChemProps`
: If `TRUE` then calculated chemical properties such as the formula and INCHIKEY are preferred over what is already present in the suspect list. For efficiency reasons it is recommended to set this to `TRUE`. See the `Validating and calculating chemical properties` section for more details.

`neutralChemProps`
: If `TRUE` then the neutral form of the molecule is considered to calculate SMILES, formulae etc. Enabling this may improve feature matching when considering common adducts (*e.g.* `[M+H]+`, `[M-H]-`). See the `Validating and calculating chemical properties` section for more details.

`onlyHits`
: If `TRUE` then all feature groups not matched by any of the suspects will be removed.

`...`
: Further arguments specified to the methods.

`amend`
: If `TRUE` then screening results will be *amended* to the original object.

`level`
: The identification level to be converted.

`out`
: The file path to the target file.

`inLevels`, `exLevels`
: A regular expression for the identification levels to include or exclude, respectively. For instance, `exLevels="4|5"` would exclude level 4 and 5 from the output file. Set to `NULL` to ignore.

Details

Besides 'full non-target analysis', where compounds may be identified with little to no prior knowledge, a common strategy is to screen for compounds with known or suspected identity. This may be a generally favorable approach if possible, as it can significantly reduce the load on data interpretation.

`screenSuspects` is used to perform suspect screening. The input `featureGroups` object will be screened for suspects by *m/z* values and optionally retention times. Afterwards, any feature groups not matched may be kept or removed, depending whether a full non-target analysis is desired.

numericIDLevel Extracts the numeric part of a given identification level (*e.g.* "3a" becomes '3').

genIDLevelRulesFile Generates a template YAML file that is used to configure the rules for automatic estimation of identification levels. This file can then be used as input for annotateSuspects.

Value

screenSuspects returns a featureGroupsScreening object, which is a copy of the input fGroups object amended with additional screening information.

Sets workflows

In a sets workflow, screenSuspects performs suspect screening for each set separately, and the screening results are combined afterwards. The sets column in the screenInfo data marks in which sets the suspect hit was found.

Suspect list format

the suspects argument for screenSuspects should be a data.frame with the following mandatory and optional columns:

- name The suspect name. Must be file-compatible. (**mandatory**)
- rt The retention time (in seconds) for the suspect. If specified the suspect will only be matched if its retention matches the experimental value (tolerance defined by the rtWindow argument). (**optional**)
- neutralMass,formula,SMILES,InChI The neutral monoisotopic mass, chemical formula, SMILES or InChI for the suspect. (data from one of these columns are **mandatory** in case no value from the mz column is available for a suspect)
- mz The ionized *m/z* of the suspect. (**mandatory** unless it can be calculated from one of the aforementioned columns)
- adduct A character that can be converted with as.adduct. Can be used to automatically calculate values for the mz column. (**mandatory** unless data from the mz column is available, the adduct argument is set or fGroups has adduct annotations)
- fragments_mz,fragments_formula One or more MS/MS fragments (specified as *m/z* or formulae, respectively). Multiple values can be specified by separating them with a semicolon (;). This data is used by annotateSuspects to report detected MS/MS fragments and calculate identification levels. (**optional**)

Matching of suspect masses

How the mass of a suspect is matched with the mass of a feature depends on the available data:

- If the suspect has data from the mz column of the suspect list, then this data is matched with the detected feature *m/z*.
- Otherwise, if the suspect has data in the adduct column of the suspect list, this data is used to calculate its mz value, which is then used like above.
- In the last case, the neutral mass of the suspect is matched with the neutral mass of the feature. Hence, either the adduct argument needs to be specified, or the featureGroups input object must have adduct annotations.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formula in the suspect list are automatically validated and calculated if missing/invalid.

The internal validation/calculation process performs the following steps:

- Validation of SMILES, INCHI, INCHIKEY and formula data (if present). Invalid entries will be set to NA.
- If `neutralChemProps=TRUE` then chemical data (SMILES, formulae etc.) is neutralized by (de-)protonation (using the `--neutralized` option of `OpenBabel`). An additional column `molNeutralized` is added to mark those molecules that were neutralized. Note that neutralization requires either SMILES or INCHI data to be available.
- The SMILES and INCHI data are used to calculate missing or invalid SMILES, INCHI, INCHIKEY and formula data. If `prefCalcChemProps=TRUE` then existing INCHIKEY and formula data is overwritten by calculated values whenever possible.
- The chemical formulae which were *not* calculated are verified and normalized. This process may be time consuming, and is potentially largely avoided by setting `prefCalcChemProps=TRUE`.
- Neutral masses are calculated for missing values (`prefCalcChemProps=FALSE`) or whenever possible (`prefCalcChemProps=TRUE`).

Note that calculation of formulae for molecules that are isotopically labelled is currently only supported for deuterium (2H) elements.

This functionality relies heavily on OpenBabel, please make sure it is installed.

Note

Both `screenSuspects` may use the suspect names to base file names used for reporting, logging etc. Therefore, it is important that these are file-compatible names. For this purpose, `screenSuspects` will automatically try to convert long, non-unique and/or otherwise incompatible suspect names.

References

OBoyle NM, Banck M, James CA, Morley C, Vandermeersch T, Hutchison GR (2011). "Open Babel: An open chemical toolbox." *Journal of Cheminformatics*, **3**(1). doi:10.1186/17582946333.

See Also

`featureGroupsScreening`

`sets-workflow` *Sets workflows*

Description

With sets workflows in **patRoon** a complete non-target (or suspect) screening workflow is performed with sample analyses that were measured with different MS methods (typically positive and negative ionization).

Details

The analyses files that were measured with a different method are grouped in *sets*. In the most typical case, there is a "positive" and "negative" set, for the positively/negatively ionized data, respectively. However, other distinctions than polarity are also possible (although currently the chromatographic method should be the same between sets). A sets workflow is typically initiated with the makeSet method. The handbook contains much more details about sets workflows.

See Also

makeSet to initiate sets workflows, workflowStepSet, the Sets workflows sections in other documentation pages and the **patRoon** handbook.

---

settings *Compounds list class for MetFrag results.*

Description

This class is derived from compounds and contains additional specific MetFrag data.

Usage

```
settings(compoundsMF)

## S4 method for signature 'compoundsMF'
settings(compoundsMF)
```

Arguments

compoundsMF	A compoundsMF object.

Details

Objects from this class are generated by generateCompoundsMetFrag

Methods (by generic)

- settings(compoundsMF): Accessor method for the settings slot.

Slots

settings A list with all general configuration settings passed to MetFrag. Feature specific items (*e.g.* spectra and precursor masses) are not contained in this list.

S4 class hierarchy

- compounds
  - compoundsMF

References

Ruttkies C, Schymanski EL, Wolf S, Hollender J, Neumann S (2016). "MetFrag relaunched: incorporating strategies beyond in silico fragmentation." *Journal of Cheminformatics*, **8**(1). doi:10.1186/s1332101601159.

See Also

`compounds` and `generateCompoundsMetFrag`

`specSimParams` *MS spectral similarity calculation parameters*

Description

Parameters relevant for calculation of similarities between mass spectra.

Usage

```
getDefSpecSimParams(...)
```

Arguments

...	optional named arguments that override defaults.

Details

For the calculation of spectral similarities the following parameters exist:

- `method` The similarity method: either `"cosine"` or `"jaccard"`.
- `removePrecursor` If `TRUE` then precursor peaks (*i.e.* the mass peak corresponding to the feature) are removed prior to similarity calculation.
- `mzWeight`,`intWeight` Mass and intensity weights used for cosine calculation.
- `absMzDev` Maximum absolute *m/z* deviation between mass peaks, used for binning spectra.
- `relMinIntensity` The minimum relative intensity for mass peaks ('`0-1`'). Peaks with lower intensities are not considered for similarity calculation. The relative intensities are called after the precursor peak is removed when `removePrecursor=TRUE`.
- `minPeaks` Only consider spectra that have at least this amount of peaks (*after* the spectrum is filtered).
- `shift` If and how shifting is applied prior to similarity calculation. Valid options are: `"none"` (no shifting), `"precursor"` (all mass peaks of the second spectrum are shifted by the mass difference between the precursors of both spectra) or `"both"` (the spectra are first binned without shifting, and peaks still unaligned are then shifted as is done when `shift="precursor"`).
- `setCombinedMethod` (**sets workflow**) Determines how spectral similarities from different sets are combined. Possible values are `"mean"`, `"min"` or `"max"`, which calculates the combined value as the mean, minimum or maximum value, respectively. `NA` values (*e.g.* if a set does not have peak list data to combine) are removed in advance.

These parameters are typically passed as a named `list` as the `specSimParams` argument to functions that do spectral similarity calculations. The `getDefSpecSimParams` function can be used to generate such parameter list with defaults.

`transformationProductsFormula-class`
Base transformation products (TP) class with formula information

---

Description

Holds information for all TPs for a set of parents, including chemical formulae.

Usage

```
## S4 method for signature 'transformationProductsFormula'
plotGraph(
  obj,
  which,
  components = NULL,
  prune = TRUE,
  onlyCompletePaths = FALSE,
  width = NULL,
  height = NULL
)
```

Arguments

`obj`	`transformationProductsFormula` derived object to be accessed
`which`	Either a `character` or `integer` vector with one or more names/indices of the parents to plot.
`components`	If specified (*i.e.* not `NULL`), a `componentsTPs` object that is used for matching the graph with screening results. The TPs that were found will be marked. See also the `prune` and `onlyCompletePaths` arguments.
`prune`	If `TRUE` and `components` is set, then pathways without *any* detected TPs are not shown (pruned). See also the `onlyCompletePaths` and `components` arguments.
`onlyCompletePaths`	If `TRUE` and `components` is set, then only pathways are shown for which *all* TPs were detected. See also the `prune` and `components` arguments.
`width, height`	Passed to `visNetwork`.

Details

This (virtual) class is derived from the `transformationProducts` base class, please see its documentation for more details. Objects from this class are returned by TP generators. More specifically, algorithms that works with chemical formulae (*e.g.* `library_formula`), uses this class to store their results. The methods defined for this class extend the functionality for the base `transformationProducts` class.

Value

plotGraph returns the result of visNetwork.

Methods (by generic)

- plotGraph(transformationProductsFormula): Plots an interactive hierarchy graph of the transformation products. The resulting graph can be browsed interactively and allows exploration of the different TP formation pathways. Furthermore, results from TP componentization can be used to match the hierarchy with screening results. The graph is rendered with **visNetwork**.

S4 class hierarchy

- transformationProducts
  - transformationProductsFormula
    - transformationProductsLibraryFormula

See Also

The base class transformationProducts for more relevant methods and generateTPs

---

transformationProductsStructure-class

Base transformation products (TP) class with structure information

---

Description

Holds information for all TPs for a set of parents, including structural information.

Usage

```
## S4 method for signature 'transformationProductsStructure'
convertToMFDB(TPs, out, includeParents = FALSE)

## S4 method for signature 'transformationProductsStructure'
filter(
  obj,
  ...,
  removeParentIsomers = FALSE,
  removeTPIsomers = FALSE,
  removeDuplicates = FALSE,
  minSimilarity = NULL,
  verbose = TRUE,
  negate = FALSE
)
```

```
## S4 method for signature 'transformationProductsStructure'
plotGraph(
  obj,
  which,
  components = NULL,
  structuresMax = 25,
  prune = TRUE,
  onlyCompletePaths = FALSE,
  width = NULL,
  height = NULL
)

## S4 method for signature 'transformationProductsStructure'
plotVenn(obj, ..., commonParents = FALSE, labels = NULL, vennArgs = NULL)

## S4 method for signature 'transformationProductsStructure'
plotUpSet(
  obj,
  ...,
  commonParents = FALSE,
  labels = NULL,
  nsets = length(list(...)) + 1,
  nintersects = NA,
  upsetArgs = NULL
)

## S4 method for signature 'transformationProductsStructure'
consensus(
  obj,
  ...,
  absMinAbundance = NULL,
  relMinAbundance = NULL,
  uniqueFrom = NULL,
  uniqueOuter = FALSE,
  labels = NULL
)
```

Arguments

`out`	The file name of the the output `MetFrag` database.
`includeParents`	Set to `TRUE` to include the parents in the database.
`obj`, `TPs`	`transformationProductsStructure` derived object to be accessed
`...`	For `filter`: Further argument passed to the base `filter method`. For `plotVenn`, `plotUpSet` and `consensus`: further (unique) `transformationProductsStructure` objects.
`removeParentIsomers`	If `TRUE` then TPs with an equal formula as their parent (isomers) are removed.

`removeTPIsomers`
: If `TRUE` then all TPs with equal formula as any sibling TPs (isomers) are removed. Unlike `removeDuplicates`, *all* TP candidates are removed (including the first match). This filter automatically sets `removeDuplicates=TRUE` so that TPs are only removed if with different structure.

`removeDuplicates`
: If `TRUE` then the TPs of a parent with duplicate structures (SMILES) are removed. Such duplicates may occur when different transformation pathways yield the same TPs. The first TP candidate with duplicate structure will be kept.

`minSimilarity`
: Minimum structure similarity ('`0-1`') that a TP should have relative to its parent. This data is only available if the `calcSims` argument to `generateTPs` was set to `TRUE`. May be useful under the assumption that parents and TPs who have a high structural similarity, also likely have a high MS/MS spectral similarity (which can be evaluated after componentization with `generateComponentsTPs`. Any values that are `NA` are removed (which only occur when a consensus was made from objects that not all have similarity information).

`verbose`
: If set to `FALSE` then no text output is shown.

`negate`
: If `TRUE` then filters are performed in opposite manner.

`which`
: Either a `character` or `integer` vector with one or more names/indices of the parents to plot.

`components`
: If specified (*i.e.* not `NULL`), a `componentsTPs` object that is used for matching the graph with screening results. The TPs that were found will be marked. See also the `prune` and `onlyCompletePaths` arguments.

`structuresMax`
: An `integer` with the maximum number of structures to plot. Setting a maximum is mainly done to avoid long times needed to construct the graph.

`prune`
: If `TRUE` and `components` is set, then pathways without *any* detected TPs are not shown (pruned). See also the `onlyCompletePaths` and `components` arguments.

`onlyCompletePaths`
: If `TRUE` and `components` is set, then only pathways are shown for which *all* TPs were detected. See also the `prune` and `components` arguments.

`width, height`
: Passed to `visNetwork`.

`commonParents`
: Only consider TPs from parents that are common to all compared objects.

`labels`
: A `character` with names to use for labelling. If `NULL` labels are automatically generated.

`vennArgs`
: A `list` with further arguments passed to **VennDiagram** plotting functions. Set to `NULL` to ignore.

`nsets, nintersects`
: See `upset`.

`upsetArgs`
: A list with any further arguments to be passed to `upset`. Set to `NULL` to ignore.

`absMinAbundance, relMinAbundance`
: Minimum absolute or relative ('`0-1`') abundance across objects for a result to be kept. For instance, `relMinAbundance=0.5` means that a result should be present in at least half of the number of compared objects. Set to '`NULL`' to ignore and keep all results. Limits cannot be set when `uniqueFrom` is not `NULL`.

`uniqueFrom` Set this argument to only retain TPs that are unique within one or more of the objects for which the consensus is made. Selection is done by setting the value of `uniqueFrom` to a `logical` (values are recycled), `numeric` (select by index) or a `character` (as obtained with `algorithm(obj)`). For `logical` and `numeric` values the order corresponds to the order of the objects given for the consensus. Set to `NULL` to ignore.

`uniqueOuter` If `uniqueFrom` is not `NULL` and if `uniqueOuter=TRUE`: only retain data that are also unique between objects specified in `uniqueFrom`.

Details

This (virtual) class is derived from the `transformationProducts` base class, please see its documentation for more details. Objects from this class are returned by TP generators. More specifically, algorithms that works with chemical structures (*e.g.* `biotransformer`), uses this class to store their results. The methods defined for this class extend the functionality for the base `transformationProducts` class.

Value

`filter` returns a filtered `transformationProductsStructure` object.

`plotGraph` returns the result of `visNetwork`.

`plotVenn` (invisibly) returns a list with the following fields:

- `gList` the `gList` object that was returned by the utilized **VennDiagram** plotting function.
- `areas` The total area for each plotted group.
- `intersectionCounts` The number of intersections between groups.

The order for the `areas` and `intersectionCounts` fields is the same as the parameter order from the used plotting function (see *e.g.* `draw.pairwise.venn` and `draw.triple.venn`).

`consensus` returns a `transformationProductsStructure` object that is produced by merging results from multiple `transformationProductsStructure` objects.

Methods (by generic)

- `convertToMFDB(transformationProductsStructure)`: Exports this object as a '`.csv`' file that can be used as a `MetFrag` local database. Any duplicate TPs (formed by different pathways or parents) will be merged based on their INCHIKEY.
- `filter(transformationProductsStructure)`: Performs rule-based filtering. Useful to simplify and clean-up the data.
- `plotGraph(transformationProductsStructure)`: Plots an interactive hierarchy graph of the transformation products. The resulting graph can be browsed interactively and allows exploration of the different TP formation pathways. Furthermore, results from TP componentization can be used to match the hierarchy with screening results. The graph is rendered with **visNetwork**.
- `plotVenn(transformationProductsStructure)`: plots a Venn diagram (using **VennDiagram**) outlining unique and shared candidates of up to five different `featureAnnotations` objects.

- `plotUpSet(transformationProductsStructure)`: Plots an UpSet diagram (using the `upset` function) outlining unique and shared TPs between different `transformationProductsStructure` objects.
- `consensus(transformationProductsStructure)`: Generates a consensus from different `transformationProductsStructure` objects. Currently this removes any hierarchical data, and all TPs are considered to originate from the same (original) parent.

Comparison between objects

The methods that compare different objects (*e.g.* `plotVenn` and `consensus`) use the INCHIKEY to match TPs between objects. Moreover, the parents between objects are matched by their name. Hence, it is *crucial* that the input parents to `generateTPs` (*i.e.* the `parents` argument) are named equally.

S4 class hierarchy

- `transformationProducts`
  - `transformationProductsStructure`
    - `transformationProductsStructureConsensus`
    - `transformationProductsCTS`
    - `transformationProductsBT`
    - `transformationProductsLibrary`

Note

`consensus`: If the `retDir` values differs between matched TPs it will be set to '`0`'. If structure similarity data is available (*i.e.* `calcSims=TRUE` to `generateTPs`) then the mean similarity is calculated.

References

Conway JR, Lex A, Gehlenborg N (2017). "UpSetR: an R package for the visualization of intersecting sets and their properties." *Bioinformatics*, **33**(18), 2938-2940. doi:10.1093/bioinformatics/btx364, http://dx.doi.org/10.1093/bioinformatics/btx364.

Lex A, Gehlenborg N, Strobelt H, Vuillemot R, Pfister H (2014). "UpSet: Visualization of Intersecting Sets." *IEEE Transactions on Visualization and Computer Graphics*, **20**(12), 1983–1992. doi:10.1109/tvcg.2014.2346248.

See Also

The base class `transformationProducts` for more relevant methods and `generateTPs`

verifyDependencies *Verifies if all dependencies are installed properly and instructs the user if this is not the case.*

Description

Verifies if all dependencies are installed properly and instructs the user if this is not the case.

Usage

```
verifyDependencies()
```

withOpt *Temporarily changes package options*

Description

This function is inspired by `withr::with_options`: it can be used to execute some code where package options are temporarily changed. This function uses a shortened syntax, especially when changing options for `patRoon`.

Usage

```
withOpt(code, ..., prefix = "patRoon.")
```

Arguments

`code`	The code to be executed.
`...`	Named arguments with options to change.
`prefix`	A `character` that will be used to prefix given option names.

Examples

```
## Not run:
# Set max parallel processes to five while performing formula calculations
withOpt(MP.maxProcs = 5, {
    formulas <- generateFormulas(fGroups, "genform", ...)
})

## End(Not run)
```

`workflowStep-class` *(Virtual) Base class for all workflow objects.*

Description

All workflow objects (*e.g.* `featureGroups`, `compounds`, etc) are derived from this class. Objects from this class are never created directly.

Usage

```
## S4 method for signature 'workflowStep'
algorithm(obj)

## S4 method for signature 'workflowStep'
as.data.table(x, keep.rownames = FALSE, ...)

## S4 method for signature 'workflowStep'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S4 method for signature 'workflowStep'
show(object)
```

Arguments

`obj`, `x`, `object`	An object (derived from) this class.
`keep.rownames`	Ignored.
`...`	Method specific arguments. Please see the documentation of the derived classes.
`row.names`, `optional`	Ignored.

Methods (by generic)

- `algorithm(workflowStep)`: Returns the algorithm that was used to generate an object.
- `as.data.table(workflowStep)`: Summarizes the data in this object and returns this as a `data.table`.
- `as.data.frame(workflowStep)`: This method simply calls `as.data.table` and converts the result to a classic a `data.frame`.
- `show(workflowStep)`: Shows summary information for this object.

Slots

`algorithm` The algorithm that was used to generate this object. Use the `algorithm` method for access.

S4 class hierarchy

- workflowStep
  - transformationProducts
    - transformationProductsStructure
      - transformationProductsStructureConsensus
      - transformationProductsCTS
      - transformationProductsBT
      - transformationProductsLibrary
    - transformationProductsFormula
      - transformationProductsLibraryFormula
    - transformationProductsLogic
  - features
    - featuresSet
    - featuresUnset
    - featuresFromFeatGroups
    - featuresConsensus
    - featuresBruker
    - featuresEnviPick
    - featuresKPIC2
    - featuresOpenMS
    - featuresSAFD
    - featuresSIRIUS
    - featuresBrukerTASQ
    - featuresXCMS
    - featuresXCMS3
  - featureGroups
    - featureGroupsSet
      - featureGroupsScreeningSet
    - featureGroupsUnset
    - featureGroupsScreening
      - featureGroupsSetScreeningUnset
    - featureGroupsBruker
    - featureGroupsConsensus
    - featureGroupsEnviMass
    - featureGroupsKPIC2
    - featureGroupsOpenMS
    - featureGroupsSIRIUS
    - featureGroupsBrukerTASQ
    - featureGroupsXCMS
    - featureGroupsXCMS3
  - components
    - componentsCamera

`workflowStepSet-class` *(Virtual) base class for sets related workflow objects*

Description

This class is the base for many sets workflows related classes. This class is virtual, and therefore never created directly.

Usage

```
## S4 method for signature 'workflowStepSet'
setObjects(obj)

## S4 method for signature 'workflowStepSet'
sets(obj)

## S4 method for signature 'workflowStepSet'
show(object)
```

Arguments

obj, object	An object that is derived from workflowStepSet.

Details

The most important purpose of this class is to hold data that is specific for a set. These *set objects* are typically objects with classes from a regular non-sets workflow (*e.g.* components, compounds), and are used by the sets workflow object to *e.g.* form a consensus. Since the set objects may contain additional data, such as algorithm specific slots, it may in some cases be of interest to access them directly with the setObjects method (described below).

Methods (by generic)

- setObjects(workflowStepSet): Accessor for the setObjects slot.
- sets(workflowStepSet): Returns the names for each set in this object.
- show(workflowStepSet): Shows summary information for this object.

Slots

setObjects A list with the *set objects* (see the Details section). The list is named with the set names.

S4 class hierarchy

- workflowStepSet
  - componentsSet
    - componentsNTSet
  - featureGroupsScreeningSet
  - compoundsSet
    - compoundsConsensusSet
  - formulasSet
    - formulasConsensusSet
  - MSPeakListsSet

Index