

Package ‘patRoön’

April 8, 2026

Type Package

Title Workflows for Mass-Spectrometry Based Non-Target Analysis

Version 3.0.0

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 7.3.3

URL <https://github.com/rickhelmus/patRoön>

BugReports <https://github.com/rickhelmus/patRoön/issues>

Encoding UTF-8

Depends R (>= 4.1.0)

SystemRequirements GNU make

Imports methods,
checkmate (>= 1.8.5),
Rcpp,
VennDiagram,
UpSetR,
stats,
utils,
parallel,
grid,
graphics,
RColorBrewer,
data.table (>= 1.15.0),
withr,
digest,

DBI,
RSQLite ($\geq 2.2.4$),
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,
bslib ($\geq 0.9.0$),
bsicons,
reactable ($\geq 0.4.1$),
magrittr,
kableExtra,
R.utils,
magick,
glue,
jsonlite,
Rdpack,
rsm,
future,
future.apply,
fs,
yaml,
keys,

backports,
 httr,
 getPass,
 lzstring,
 constructive,
 signal,
 MBA

Suggests RDCOMClient,

metfRag,
 enviPick,
 nontarget,
 RAMClustR,
 cliqueMS,
 KPIC,
 MetaClean,
 MetaCleanData,
 testthat,
 rlang,
 vdiffR,
 patRoonaData,
 devtools,
 covr,
 DiagrammeR,
 DiagrammeRsvg,
 rsvg,
 pkgload,
 splashR,
 MS2Quant,
 MS2Tox,
 ChemmineR,
 fmcsR,
 reticulate,
 Rmstoolkitlib

LinkingTo Rcpp,
 rapidjsonr

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first

feature*, screening, compounds, components, tp, formulas, pred, mspeaklists, project-tool, report

RdMacros Rdpack

Collate 'RcppExports.R'

'generics.R'
 'cache.R'
 'main.R'
 'workflow-step.R'
 'TP.R'

'TP-structure.R'
'TP-CTS.R'
'TP-ann_comp.R'
'TP-formula.R'
'TP-ann_form.R'
'TP-biotransformer.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'
'utils-bruker.R'
'deprecated-formulas-bruker.R'
'deprecated-mspeaklists-bruker.R'
'deprecated-mspeaklists-mzr.R'
'deprecated.R'
'utils-IPO.R'
'doe-optimizer.R'
'eic_eim_gui.R'
'feature_groups-ADT.R'
'feature_groups-bruker.R'
'feature_groups-comparison.R'
'feature_groups-envimass.R'
'feature_groups-filter.R'
'feature_groups-greedy.R'
'feature_groups-ims.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-table.R'
'feature_groups-tasq.R'
'feature_groups-xcms.R'
'feature_groups-xcms3.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-piek.R'
'features-safd.R'
'features-sirius.R'
'features-table.R'
'features-tasq.R'

'features-xcms.R'
'features-xcms3.R'
'formulas-genform.R'
'formulas-set.R'
'formulas-sirius.R'
'limits.R'
'msdata.R'
'mslibrary-json.R'
'mslibrary-msp.R'
'mspeaklists-set.R'
'multi-process-classic.R'
'multi-process-future.R'
'multi-process.R'
'peaks.R'
'project-tool-analyses.R'
'project-tool-annotation.R'
'project-tool-features.R'
'project-tool-gen.R'
'project-tool-general.R'
'project-tool-pre_treatment.R'
'project-tool-report.R'
'project-tool-tp.R'
'project-tool-utils.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-ims.R'
'utils-features-sets.R'
'utils-features.R'
'utils-formulas.R'
'utils-mol.R'
'utils-msdata.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	10
adduct-class	12
adduct-utils	13
analysis-information	15
analysisinfo-dataframe	17
assignMobilities_comp	20
assignMobilities_feat	21
assignMobilities_susp	26
bruker-utils	29
C3SDBAdducts	31
caching	32
CCS-Conversion	33
check-GUI	34
cluster-params	37
components-class	38
componentsClust-class	44
componentsIntClust-class	47
componentsNT-class	49
componentsSpecClust-class	51
componentsTPs-class	51
compounds-class	54
compounds-cluster	63
compoundsCluster-class	64
compoundScorings	68
compoundsMF-class	68
compoundsSIRIUS-class	69
defaultOpenMSAdducts	70
EIXParams	71
feature-filtering	72
feature-optimization	78
feature-plotting	82
feature-quality	91
feature-table	92
featureAnnotations-class	95
featureGroups-class	101
featureGroups-compare	116
featureGroupsComparison-class	118
featureGroupsScreening-class	120
features-class	124
findFeatures	132
findFeaturesBruker	133

findFeaturesEnviPick	134
findFeaturesKPIC2	135
findFeaturesOpenMS	136
findFeaturesPiek	139
findFeaturesSAFD	145
findFeaturesSIRIUS	147
findFeaturesXCMS	148
findFeaturesXCMS3	149
formulas-class	150
formulaScorings	158
formulasSIRIUS-class	158
generateComponents	159
generateComponentsCAMERA	160
generateComponentsCliqueMS	162
generateComponentsIntClust	165
generateComponentsNontarget	167
generateComponentsOpenMS	169
generateComponentsRAMClustR	172
generateComponentsSpecClust	175
generateComponentsTPs	177
generateCompounds	180
generateCompoundsLibrary	182
generateCompoundsMetFrag	185
generateCompoundsSIRIUS	191
generateFormulas	195
generateFormulasGenForm	197
generateFormulasSIRIUS	203
generateMSPeakLists	207
generateTPs	209
generateTPsAnnComp	210
generateTPsAnnForm	214
generateTPsBioTransformer	216
generateTPsCTS	220
generateTPsLibrary	222
generateTPsLibraryFormula	225
generateTPsLogic	228
generics	229
genFormulaTPLibrary	238
getBGMSMSPeaks	240
getCCSParams	242
getDefAvgPListParams	244
getDefPeakParams	246
getDefTPStructParams	249
getEICs	250
getFCParams	251
getIMSMATCHParams	252
getIMSRANGEParams	253
getMSFileFormats	253

getMSFileTypes	254
groupFeatures	254
groupFeaturesGreedy	255
groupFeaturesKPIC2	257
groupFeaturesOpenMS	258
groupFeaturesSIRIUS	260
groupFeaturesXCMS	261
groupFeaturesXCMS3	262
id-conf	264
importFeatureGroups	270
importFeatureGroupsBrukerPA	271
importFeatureGroupsBrukerTASQ	272
importFeatureGroupsEnviMass	273
importFeatureGroupsKPIC2	274
importFeatureGroupsTable	275
importFeatureGroupsXCMS	276
importFeatureGroupsXCMS3	277
importFeatures	278
importFeaturesEnviMass	279
importFeaturesKPIC2	280
importFeaturesTable	281
importFeaturesXCMS	283
importFeaturesXCMS3	284
installC3SDB	285
installTIMSCONVERT	285
kplic2-conv	286
launchEICGUI	287
limits	288
loadMSLibrary	290
loadMSLibraryMoNAJSON	291
loadMSLibraryMSP	294
makeSet	297
MSConversion	299
msdata	304
MSLibrary-class	307
MSPeakLists-class	313
newProject	323
optimizationResult-class	323
pred-aggr-params	326
pred-quant	327
pred-tox	332
printPackageOpts	335
reporting	336
reporting-legacy	340
retDir	344
sets-workflow	344
specSimParams	345
suspect-screening	346

TPLogicTransformations	350
transformationProducts-class	351
transformationProductsAnnComp-class	354
transformationProductsAnnForm-class	355
transformationProductsFormula-class	357
transformationProductsStructure-class	358
verifyDependencies	363
withOpt	363
workflowStep-class	364
workflowStepSet-class	367
xcms-conv	368

Index	371
--------------	------------

patRoön-package	<i>Workflow solutions for mass-spectrometry based non-target analysis.</i>
-----------------	--

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:

- `patRoön.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.
- `patRoön.cache.fileName`: a character specifying the name of the cache file (default is 'cache.sqlite').
- `patRoön.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.
- `patRoön.MS.backends`, `patRoön.MS.preferIMS`, `patRoön.path.TDFSDK`: Options related to the [raw data interface](#).
- `patRoön.threads`: The number of threads to be used for parallelization. This is currently only used by the [raw data interface](#) and when the `piek` algorithm is used for [peak detection](#).
- `patRoön.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 '`patRoön.MP.method="classic"`'.

See Also

Useful links:

- <https://github.com/rickhelmus/patRoan>
- Report bugs at <https://github.com/rickhelmus/patRoan/issues>

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	<p>A character that specifies the source format.</p> <p>"generic" is an internally used generic format that supports full textual conversion. Examples: "[M+H]+", "[2M+H]+", "[M+3H]3+".</p> <p>"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.</p> <p>"genform" and "metfrag" support fixed types of adducts which can be obtained with the GenFormAdducts and MetFragAdducts functions, respectively.</p> <p>"openms" is the format used by the MetaboliteAdductDecharger tool.</p> <p>"cliquems" is the format used by cliqueMS.</p>
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	<p>An adduct object (or something that can be converted to it with as.adduct).</p> <p>Examples: "[M-H]-", "[M+Na]+".</p>

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(
  fromRaw = NULL,
  fromCentroid = NULL,
  fromProfile = NULL,
  fromIMS = NULL,
  convCentroid = NULL,
  convProfile = NULL,
  convIMS = NULL,
  ...
)

generateAnalysisInfoFromEnviMass(path)
```

Arguments

fromRaw, fromCentroid, fromProfile, fromIMS	One or more file paths that should be used for finding analyses that are stored as raw, centroided, profile or IMS data, respectively (see details below). Set to NULL to skip file detection for a particular file type.
convCentroid, convProfile, convIMS	These arguments specify the MS file conversion destination paths for centroided, profile and IMS data, respectively. These paths are used for those analyses for which no file with a particular file type could be found in the directories specified by the respective from* arguments. Set to NULL to not set any destination directory. If multiple paths are specified then these will be recycled to fill the table rows.
...	Any other columns that should be added to the analysis information table, such as replicate and blank. The arguments specified by ... should be named. Vectors are recycled to the number of rows of the table.
path	The path of the enviMass project.

Details

In **patRoön** 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. The analysis information should be a `data.frame` or `data.table` with a set of mandatory and optional columns (described below).

`generateAnalysisInfo` is an utility function that automatically generates analysis information. It scans given directories for analysis files, and uses this to automatically fill in the `analysis` and `path_*` columns. This function automatically groups together analyses that are stored with different file types and formats (see further details below).

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

Value

`generateAnalysisInformation` returns a `data.frame` with automatically generated analysis information.

Mandatory analysis information columns

The following columns should be present in the analysis information:

- `path_raw`, `path_centroid`, `path_profile`, `path_ims` Specifies the directory path for the raw, centroided, profile and IMS data, respectively. See below for more details. At least one column should not be empty for each row.
- `analysis` the file name **without** extension and without directory path. Must be **unique** across all table rows.
- `replicate` name of the *replicate*. Used to group analyses together that are replicates of each other. Thus, the `replicate` column for all analyses considered to be belonging to the same replicate should have an equal (but unique) value. Used for *e.g.* averaging and `filter`.
- `blank` all analyses within this replicate are used by the `featureGroups` method of `filter` for blank subtraction. Multiple entries can be entered by separation with a comma. May be empty ("") if no blank subtraction is desired.

Analysis paths, file types and file formats

Depending on the workflow step, different *file types* for the same analysis may be required.

- `raw` Specifies the directory to raw HRMS files (*e.g.* `‘.raw’`, `‘.d’`). This is used by *e.g.* `conversion of raw MS data` and the `OpenTIMS backend`.
- `centroid` Specifies the directory to centroided and exported HRMS files (`‘.mzML’`, `‘.mzXML’`). These files are required by most feature finding algorithms.
- `profile` Specifies the directory to exported but not centroided (*i.e.* `profile`) HRMS data files (`‘.mzML’`, `‘.mzXML’`). This is currently only used by `findFeaturesSAFD`.
- `ims` Specifies the directory to exported IMS-HRMS data (`‘.mzML’`). This is required in IMS workflows, unless raw IMS-HRMS data is directly loaded with the `OpenTIMS backend`. See *e.g.* `assignMobilities` for more details.

Some workflows may require multiple *file formats* for a same *file type*. In this case, the file formats should be stored within the same directory specified by the respective `path_*` column. For instance, if feature finding algorithms from `OpenMS` and `enviPick` are mixed then centroided `‘.mzML’` and

‘.mzXML’ files are needed, and files with both file formats must be stored in the directory specified by `path_centroid`.

If non-raw data files are not yet present and should be exported by [MS file conversion](#), then `path_centroid`, `path_profile` and `path_ims` should specify the desired destination paths of the converted files.

Optional columns and sample metadata

The following columns may need to be present:

- `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. This is used by the `as.data.table` method if `regression=TRUE`. As of **patRoan** version 3.0, any other column than “conc” can be used by setting its name with the `regression` argument.
- `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.

Any other columns that are present will be added to the features and featureGroups objects as metadata. This metadata can be used *e.g.* in various plotting and data subsetting functions.

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,
  groupBy = NULL,
  showLegend = TRUE,
```

```
    xlim = NULL,
    ylim = NULL,
    ...
)

## S4 method for signature 'data.frame'
plotBPCs(
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  groupBy = NULL,
  showLegend = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'data.table'
plotTICs(
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  groupBy = NULL,
  showLegend = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'data.table'
plotBPCs(
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  groupBy = NULL,
  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.
groupBy	Specifies how results are grouped in the plot. Should be a name of a column in the analysis information table which is used to make analysis groups (<i>e.g.</i> "replicate"), or "fGroups" to group by feature group. Set to NULL for no grouping.
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.
- `plotBPCs(data.table)`: Plots the BPCs of the analyses.

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by these functions to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

Author(s)

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

assignMobilities_comp *Assign IMS data to a compounds object.*

Description

Assigns ion mobility and CCS values to the candidates in a [compounds](#) object.

Usage

```
## S4 method for signature 'compounds'
assignMobilities(
  obj,
  fGroups,
  IMS = TRUE,
  from = NULL,
  matchFromBy = "InChIKey1",
  overwrite = FALSE,
  adduct = NULL,
  CCSParams = NULL,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE,
  virtualenv = "patRoon-C3SDB"
)

## S4 method for signature 'compoundsSet'
assignMobilities(obj, fGroups, IMS = TRUE, from = NULL, ...)
```

Arguments

obj	The compounds object for which IMS assignments should be performed.
fGroups	The featureGroups object for which the compound candidates (obj) were calculated.
IMS	(IMS workflow) Specifies which feature groups are considered for IMS assignments in IMS workflows. The following options are valid: <ul style="list-style-type: none"> • "both": Selects IMS and non-IMS features. • "maybe": Selects non-IMS features and IMS features without assigned IMS precursor. • FALSE: Selects only non-IMS features. • TRUE: Selects only IMS features.
from, matchFromBy, overwrite, CCSParams, prefCalcChemProps, neutralChemProps, virtualenv	Passed to the method for suspects
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.

... (**sets workflow**) Further arguments passed to the non-sets workflow method.

Details

The assignMobilities method for [compounds](#) is used to (1) add predicted or known IMS data to annotation candidates and (2) convert (previously added) mobility <=> CCS values. Internally, the [assignMobilities method for suspects](#) is used to perform these operations, please see its documentation for more details.

If both adduct specific and non-adduct specific mobility and CCS data is available (and not NA), then the non-adduct specific data is assigned to the compound candidate. Otherwise, data corresponding to the adduct argument or the adduct assigned to the feature group is taken. The [filter](#) method can be used to filter out candidates with deviating IMS data.

Sets workflows

In a [sets workflow](#) the calculations are performed and stored independently for each set, as adducts, charges and m/z values typically differ.

The from argument may be a list that specifies the from values for each set. This is primarily intended when tables are used to set from.

Note

SIRIUS does currently not report INCHIKEY values, hence, matchFromBy="InChIKey" is not supported in this case.

If compound annotation is performed with [generateCompoundsMetFrag](#) with database="pubchemlite" then CCS data is already added, provided the local database has it. If you want to overwrite this data, set overwrite=TRUE.

See Also

The [assignMobilities method for suspects](#).

assignMobilities_feat *Assign Ion Mobility and CCS values to features*

Description

Various approaches to assign mobilities to features and perform CCS conversions.

Usage

```
## S4 method for signature 'featureGroups'
assignMobilities(
  obj,
  mobPeakParams = NULL,
  chromPeakParams = NULL,
  EIMParams = getDefEIMParams(),
  EICParams = getDefEICParams(),
  peakRTWindow = defaultLim("retention", "narrow"),
  fallbackEIC = TRUE,
  calcArea = "integrate",
  mobWindow = defaultLim("mobility", "medium"),
  scoreWeights = c(mobility = 1, intensity = 1),
  CCSParams = NULL,
  parallel = "maybe"
)

## S4 method for signature 'featureGroupsSet'
assignMobilities(
  obj,
  mobPeakParams = NULL,
  chromPeakParams = NULL,
  EIMParams = getDefEIMParams(),
  EICParams = getDefEICParams(),
  peakRTWindow = defaultLim("retention", "narrow"),
  fallbackEIC = TRUE,
  calcArea = "integrate",
  mobWindow = defaultLim("mobility", "medium"),
  scoreWeights = c(mobility = 1, intensity = 1),
  CCSParams = NULL,
  parallel = "maybe"
)

## S4 method for signature 'featureGroupsScreening'
assignMobilities(
  obj,
  mobPeakParams = NULL,
  chromPeakParams = NULL,
  EIMParams = getDefEIMParams(),
  EICParams = getDefEICParams(),
  peakRTWindow = defaultLim("retention", "narrow"),
  fallbackEIC = TRUE,
  calcArea = "integrate",
  mobWindow = defaultLim("mobility", "medium"),
  scoreWeights = c(mobility = 1, intensity = 1),
  CCSParams = NULL,
  parallel = "maybe",
  fromSuspects = FALSE,
```

```

    IMSMatchParams = NULL
  )

## S4 method for signature 'featureGroupsScreeningSet'
assignMobilities(
  obj,
  mobPeakParams = NULL,
  chromPeakParams = NULL,
  EIMParams = getDefEIMParams(),
  EICParams = getDefEICParams(),
  peakRTWindow = defaultLim("retention", "narrow"),
  fallbackEIC = TRUE,
  calcArea = "integrate",
  mobWindow = defaultLim("mobility", "medium"),
  scoreWeights = c(mobility = 1, intensity = 1),
  CCSParams = NULL,
  parallel = "maybe",
  fromSuspects = FALSE,
  IMSMatchParams = NULL
)

```

Arguments

obj	A featureGroups (derived) object to which IMS data should be assigned.
mobPeakParams	A list with peak detection parameters for mobility peaks, generated with getDefPeakParams . Set to NULL to skip detection of mobility peaks.
chromPeakParams	A list with peak detection parameters for the re-integration of IMS features, generated with getDefPeakParams . Set to NULL to skip re-integration by peak detection.
EIMParams, EICParams	Parameters to be used for the generation of EIMs (for mobility peak detection) and EICs (for IMS feature re-integration), generated by getDefEIMParams and getDefEICParams .
peakRTWindow	The retention time tolerance (in seconds) for detected peaks to be used for re-integration (Step 3, see Mobility assignment). This should be kept low to ensure no closely eluting features from other compounds are used. Note that peaks with retention times outside the RT window of the IMS precursor are always ignored.
fallbackEIC	Set to TRUE to allow updating intensities and areas from raw EICs in Step 3 (see Mobility assignment).
calcArea	controls how the area is calculated when updating from raw EIC data (see fallbackEIC): "integrate" (performs area integration) or "sum" (sums up all intensity data points). Note: these methods may be different than what is used by the feature detection of the IMS precursor or the peak detection algorithm used for re-integration, hence, comparing areas should be done with care.
mobWindow	The mobility tolerance window.

scoreWeights	A numeric with the weights used by greedy grouping . NOTE: only the mobility and intensity weights are used. See the documentation of groupFeaturesGreedy for more details on the scoring and grouping.
CCSParams	A list with parameters for mobility <--> CCS conversion. See getCCSParams for details and to make such parameter lists. Set to NULL to skip conversions.
parallel	If set to TRUE then code is executed in parallel through the future package. Please see the parallelization section in the handbook for more details. Alternatively, set parallel="maybe" to disable parallelization if this normally will not
fromSuspects	If TRUE then mobilities values are directly copied from suspect list data (if available). See the Post mobility assignment section.
IMSMatchParams	(IMS workflow) A list with parameters to be used for matching IMS data. See getIMSMatchParams for details and how to make such a parameter list.

Details

The assignMobilities method function for features is used (1) to assign Ion Mobility values and (2) calculate CCS values from these mobilities.

In **patRoorn**, two approaches are supported to assign mobilities to features: *direct* and *post* assignment (DMA and PMA). With the DMA the mobility values are directly assigned during feature detection. This is currently only supported by the [piek](#) and [greedy](#) algorithms or by [importing feature data](#). With PMA, the mobility values are assigned after feature detection and grouping (and possibly other steps such as filtering). Thus, the PMA approach is supported by all available feature detection algorithms in **patRoorn**. The PMA approach is further described below. Only the CCS conversion functionality of assignMobilities should be used in PMA mobility assignment workflows.

The assignment of CCS values is controlled by the CCSParams argument (see above).

In suspect screening workflows assignMobilities also assigns reference mobility and CCS values to suspect hits, and can filter hits if IMSMatchParams is set. This is similarly performed as [screenSuspects](#), please see its documentation for more details.

Post mobility assignment

The *post* assignment of mobilities occurs in the following steps:

1. Extracted ion mobilograms (EIMs) are generated for all features and subjected to automatic peak detection to obtain *mobility peaks*.
2. The detected mobility peaks in an EIM are then used to form *IMS features*. These features inherit their LC-MS properties (RT, *m/z*, etc) from the corresponding *IMS precursor*, *i.e.* the feature for which the EIM was created. The mobility peak centroids and ranges are used to assign IMS data to the IMS features. Multiple mobility peaks within the same EIM result in multiple IMS features, and each are linked to the same IMS precursor. The linkage is especially useful to keep a relation between *e.g.* protomers.
3. LC-MS properties such as the area, intensity and RTs are (optionally) updated by re-integration of detected peaks from mobility filtered extracted ion chromatograms. If peak detection is disabled or fails, then the intensity and areas can be estimated directly from raw EIC data (fallbackEIC argument).

4. Any IMS features that could not be re-integrated (either by peak detection or EIC fallback) are removed.
5. The feature grouping is updated: the IMS features with close mobilities (defined by mobWindow) within a feature group are split-off into new feature groups and linked to the original *IMS precursor* feature group. This is performed by the [greedy grouping algorithm](#). LC-MS properties and most other data such as feature group qualities and scores ([calculatePeakQualities](#)), adduct annotations (e.g. [selectIons](#)), predicted concentrations and toxicities ([calculateConcs](#) and [calculateTox](#)) and internal standards for intensity normalization ([normInts](#)) are copied from the IMS precursors to the IMS feature groups.

Note that re-running assignMobilities will first remove any existing IMS features.

Suspect screening workflows: In suspect screening workflows the fromSuspects arguments can be set to alternatively perform mobility assignment directly from the suspect list data (replacing Steps 1-2). The feature mobility is simply assigned from the suspect data and the mobility range is derived from the mobWindow argument. Relationships with IMS precursors (Step 2) are similarly formed. An advantage of this approach is that no mobility peak detection is needed, which may be useful for low intensity features where this could be difficult. Setting fromSuspects=TRUE is primarily intended for workflows where (1) the mobility of a suspect is accurately known up-front or (2) IMS data should only be used as a rough filtering step for feature data. In the latter case accurate feature mobility assignment is not of interest and the suspect IMS data is typically not accurately known (e.g. predicted), hence, for these workflows the tolerance specified by mobWindow should be increased.

With fromSuspects=TRUE no mobility peak detection is performed, hence, the actual presence of the feature is only verified in Step 3. For this reason, falling back to EIC data (fallbackEIC argument) is never performed for IMS features from suspects, and chromPeakParams must always be defined to allow chromatographic peak detection.

If both fromSuspects and mobPeakParams are set, regular mobility assignment (Steps 1-2) is performed for features without suspect hit. If multiple suspects were assigned to a feature group then suspect data is *never* used to form IMS features.

IMS features

The features (and feature groups) with IMS properties are referred to *IMS features* (and *IMS feature groups*). These are referred to *orphans* if their link to the original IMS precursor is removed (in *post* workflows, see previous section) or non-existent (*direct* workflows). The formation of orphans in *post* workflows typically occurs by removal of IMS precursors by subsetting or filtering operations.

Most data-processing functionality, such as subsetting, plotting, filtering, etc., allows to selectively operate either on the IMS features, their IMS precursors, both or either the precursor or orphans (controlled by the IMS argument to the corresponding functions).

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by assignMobilities to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

Sets workflows

Mobility assignment in a [sets workflow](#) is equivalent to non-sets workflows. However, currently there is no (known) way to relate mobilities across MS polarities (+/- mode). Thus, IMS features will always be formed for each set separately. However, *IMS precursors* will still be grouped across polarities (like a non-IMS workflow) and their links to IMS features can therefore be used to relate IMS features across MS polarities.

assignMobilities_susp *Assign IMS data to suspects*

Description

Adds calculated mobility and/or CCS data to a suspect list.

Usage

```
## S4 method for signature 'data.table'
assignMobilities(
  obj,
  from = NULL,
  matchFromBy = "InChIKey1",
  overwrite = FALSE,
  adducts = c("[M+H]+", "[M-H]-", NA),
  predictAdductOnly = TRUE,
  CCSParams = NULL,
  prepareChemProps = TRUE,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE,
  virtualenv = "patRoos-C3SDB"
)

## S4 method for signature 'data.frame'
assignMobilities(obj, ...)
```

Arguments

- | | |
|------|--|
| obj | The suspect list to which the mobility and/or CCS data should be added. Should be a <code>data.frame</code> or <code>data.table</code> . |
| from | Specifies from where IMS data is added to the suspect list. This can be the following: <ul style="list-style-type: none">• "pubchemlite": CCS data is matched from predicted values of the PubChemLite database. Note: this requires a local copy of the CCS amended PubChemLite database (see the Handbook for more details), which is automatically installed by patRoosExt.• "c3sdb": Uses the C3SDB Python package to predict CCS values. This requires a local installation of C3SDB, e.g. performed with installC3SDB. |

	<ul style="list-style-type: none"> • A <code>data.table</code> or <code>data.frame</code> to which IMS data is matched. Should contain the column defined by <code>matchFromBy</code> and columns storing (non-)adduct specific mobility/CCS columns (see Details). • <code>NULL</code>: No IMS data is added to the suspect list. <p>Any NA values in <code>from</code> are ignored.</p>
<code>matchFromBy</code>	<p>Which column should be used to match the IMS data from <code>from</code> and suspects. Valid options are <code>"InChIKey"</code>, <code>"InChIKey1"</code> (first block <code>InChIKey</code>), <code>"InChI"</code>, <code>"SMILES"</code>, <code>"name"</code>. However, this also depends on which columns are available in either of the data sources. <code>InChIKey1</code> values are automatically calculated from <code>InChIKeys</code>, if possible.</p> <p>Matching by <code>InChIKey1</code> is recommended by default to allow tolerance between different datasources. Note that most compound annotation algorithms also match by <code>InChIKey1</code> and current IMS resolving power is generally insufficient to distinguish the different stereoisomers/tautomers specified by the full <code>InChIKey</code>.</p>
<code>overwrite</code>	Set to <code>TRUE</code> to overwrite any existing suspect IMS data with data from <code>from</code> .
<code>adducts</code>	<p>A character with the adduct(s) to consider for assigning mobility data to suspects and mobility \leftrightarrow CCS conversions. This may be limited by what is available in the data source specified by <code>from</code> (see C3SDBAdducts for <code>from="c3sdb"</code>). Inclusion of NA in <code>adducts</code> allows the use of non-adduct specific values (see Details).</p> <p>The value for <code>adducts</code> is automatically expanded by the adducts specified in the <code>adduct</code> column of the suspect list. Hence, <code>adducts</code> can be empty (<code>character()</code>) if no calculations for other adducts are desired.</p>
<code>predictAdductOnly</code>	If <code>from="c3sdb"</code> and <code>predictAdductOnly=TRUE</code> then only predictions are performed for the adduct specified in the <code>adduct</code> column in the suspect list (if present).
<code>CCSParams</code>	A list with parameters for mobility \leftrightarrow CCS conversion. See getCCSParams for details and to make such parameter lists. Set to <code>NULL</code> to skip conversions.
<code>prepareChemProps</code>	Set to <code>TRUE</code> to perform chemical property calculation and validation on the suspect list (described below).
<code>prefCalcChemProps</code>	If <code>TRUE</code> then calculated chemical properties such as the formula and <code>INCHIKEY</code> are preferred over what is already present in the suspect data (if <code>prepareChemProps=TRUE</code>) and from data (if a table). For efficiency reasons it is recommended to set this to <code>TRUE</code> . See the Validating and calculating chemical properties section for more details.
<code>neutralChemProps</code>	If <code>TRUE</code> then the neutral form of the molecule is considered to calculate <code>SMILES</code> , formulae etc. Enabling this may improve feature matching when considering common adducts (e.g. <code>[M+H]⁺</code> , <code>[M-H]⁻</code>). See the Validating and calculating chemical properties section for more details.
<code>virtualenv</code>	The virtual Python environment in which C3SDB is installed. This is passed to <code>reticulate::use_virtualenv</code> . Set to <code>NULL</code> to skip this and not setup the environment.

... Arguments passed to `data.table` method.

Details

The `assignMobilities` method for suspect lists is used to (1) add IMS data to suspects from predictions or library data and (2) convert (previously added) mobility \leftrightarrow CCS values. These steps are controlled by the `from` and `CCSParms` arguments, respectively.

Mobility and CCS values assigned in the suspect list are either adduct specific or not. Adduct specific values are preferred, as the 'correct' value can be automatically selected during suspect screening based on the adduct assigned to the feature (or passed as the `adduct` argument to `screenSuspects`). The non-adduct specific values are typically used when the corresponding adduct for the mobility/CCS value is unknown (or not of interest). These values get precedence over adduct specific values. The adduct specific values are stored in `mobility_<adduct>` and `CCS_<adduct>` columns, where `<adduct>` is the adduct name (e.g. `[M+H]+`, `[M-H]-`). The mobility and CCS columns store any non-adduct specific values. The `adducts` argument ultimately defines the use of adduct and non-adduct specific values.

The mobility \leftrightarrow CCS conversions occur both ways, i.e. missing CCS values will be converted from mobility values and *vice versa*. If adduct specific values are converted then the charge value used for these calculations is taken from the corresponding adduct. For non-adduct specific values the charge is taken from the adduct specified in suspect list if present, or from the default charge specified in `CCSParms` otherwise.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formulae in the suspect data (if `prepareChemProps=TRUE`) and from data (if a table) 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

- 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.
- Schymanski EL, Kondić T, Neumann S, Thiessen PA, Zhang J, Bolton EE (2021). “Empowering large chemical knowledge bases for exposomics: PubChemLite meets MetFrag.” *Journal of Cheminformatics*, **13**(1). ISSN 1758-2946, doi:10.1186/s13321021004890, <http://dx.doi.org/10.1186/s13321-021-00489-0>.
- Elapavalore A, Ross DH, Grouès V, Aurich D, Krinsky AM, Kim S, Thiessen PA, Zhang J, Dodds JN, Baker ES, Bolton EE, Xu L, Schymanski EL (2025). “PubChemLite Plus Collision Cross Section (CCS) Values for Enhanced Interpretation of Nontarget Environmental Data.” *Environmental Science & Technology Letters*, **12**(2), 166–174. ISSN 2328-8930, doi:10.1021/acs.estlett.4c01003, <http://dx.doi.org/10.1021/acs.estlett.4c01003>.
- Ross DH, Cho JH, Xu L (2020). “Breaking Down Structural Diversity for Comprehensive Prediction of Ion-Neutral Collision Cross Sections.” *Analytical Chemistry*, **92**(6), 4548–4557. ISSN 1520-6882, doi:10.1021/acs.analchem.9b05772, <http://dx.doi.org/10.1021/acs.analchem.9b05772>.

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)

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

getDACalibrationError(anaInfo)

addDAEIC(
  analysis,
  path,
  mz,
  mzWindow = defaultLim("mz", "medium"),
  ctype = "EIC",
  mtype = "MS",
  polarity = "both",
  bgsubtr = FALSE,
```

```

    fragpath = "",
    name = NULL,
    hideDA = TRUE,
    close = FALSE,
    save = close
)

addAllDAEICs(
  fGroups,
  mzWindow = defaultLim("mz", "medium"),
  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 addAllDAEICs: 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.

`recalibrateDAFiles` 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](#)

C3SDBAdducts

Returns the adducts supported by C3SDB

Description

Returns the adducts supported by the **C3SDB** Python package.

Usage

`C3SDBAdducts()`

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

<code>...</code>	Arguments/objects to be used for hashing.
<code>checkDT</code>	logical, set to TRUE with (a list with) <code>data.tables</code> to ensure reproducible hashing. Otherwise can be set to FALSE to improve performance.
<code>length</code>	Maximum file length to hash. Passed to digest .
<code>category</code>	The category of the object to be cached.
<code>hashes</code>	A character with one more hashes (<i>e.g.</i> obtained with <code>makeHash</code>) of the objects to be loaded.
<code>dbArg</code>	Alternative connection to database. Default is NULL and uses the cache options as defined by <code>'patRoan.cache.fileName'</code> . Mainly used internally to improve performance.
<code>simplify</code>	If TRUE and <code>length(hashes)==1</code> then the returned data is returned directly, otherwise the data is in a list.
<code>fixDTs</code>	Should be TRUE if cached data consists of (nested) <code>data.tables</code> . Otherwise can be FALSE to speed up loading.
<code>data</code>	The object to be cached.
<code>hash</code>	The hash string of the object to be cached (<i>e.g.</i> obtained with <code>makeHash</code>).
<code>what</code>	This argument describes what should be done. When <code>what = NULL</code> this function will list which tables are present along with an indication of their size (database rows). If <code>what = "all"</code> then the complete file will be removed. Otherwise, <code>what</code> 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 <code>patRoam.cache.fileName</code> 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.

CCS-Conversion

Conversion between mobility and CCS

Description

Utility functions to convert between mobility and CCS data.

Usage

```
convertMobilityToCCS(mobility, mz, CCSPParams, charge = NULL)
```

```
convertCCSToMobility(ccs, mz, CCSPParams, charge = NULL)
```

Arguments

mobility, ccs	A numeric vector with mobility or CCS values that should be converted.
mz	A numeric vector with the m/z values that map to the input mobility or CCS values.
CCSPParams	A list with parameters for mobility \leftrightarrow CCS conversion. See getCCSPParams for details and to make such parameter lists. Set to NULL to skip conversions.
charge	A numeric vector with the ion charges that map to the input mobility or CCS data. Will be recycled if necessary. If NULL then the charge configured in CCSPParams is used.

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(),
  EIMParams = getDefEIMParams(),
  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 = defaultLim("retention", "narrow"),
  mzWindow = defaultLim("mz", "narrow"),
  mobWindow = defaultLim("mobility", "narrow"),
  overwrite = FALSE
)

## S4 method for signature 'featureGroups'
```

```

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

getMCTrainData(fGroups, session)

predictCheckFeaturesSession(fGroups, session, model = NULL, overwrite = FALSE)

```

Arguments

fGroups	A featureGroups object. This should be the 'new' object for <code>importCheckFeaturesSession</code> 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.
EIMParams	A named list with parameters used for extracted ion mobilogram (EIM) creation. See the EIM 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, mzWindow, mobWindow	The retention time (seconds), m/z and mobility window (if present) 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::getPredictions` function. The dataframe has the four columns: EIC, Pred_Class, Pred_Prob_Pass, Pred_Prob_Fail.

Use of raw HRMS data

The [raw data interface](#) of **patRoan** is used by these functions to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

Note

The `topMost` and `topMostByReplicate` [EIC parameters](#) 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.

cluster-params	<i>Clustering parameters</i>
----------------	------------------------------

Description

Parameters for clustering data such as mass spectra and mobilograms.

Details

Different functionality within **patRoan** uses clustering to group similar data together, for instance, to average mass spectra. A fast C++ backend based on **Rcpp** is used to perform the clustering.

The clustering can be configured by the method and window parameter. The following clustering methods are available:

- "hclust": uses hierarchical clustering to find similar data points (using **hclust-cpp**, which is based on the **fastcluster** package).
- "distance_point": uses a maximum distance between adjacent sorted data points to form clusters.
- "distance_mean": uses a maximum distance between the mean of the current cluster and the next sorted data point to form clusters.
- "bin": uses a simple binning approach to cluster data points.

The hclust method may give more accurate results and was the default prior to **patRoan 3.0**, but is more computationally demanding and generally unsuitable for IMS workflows due to excessive use of RAM. The distance_* methods are now default and suit most cases.

The window parameter defines the clustering tolerance. For method="hclust" this corresponds to the cluster height, for method="distance_*" methods this value sets the maximum distance between compared data and for method="bin" it corresponds to the bin width. Too small windows will prevent clustering close data points (e.g. resulting in split mass peaks in averaged spectra), whereas too big windows may cluster unrelated data points together (e.g. resulting in mass inaccuracies).

Source

Averaging of mass spectra was originally based on algorithms from the **msProcess** R package (now archived on CRAN).

References

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 (2026). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.1, <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.

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.

components-class	<i>Component class</i>
------------------	------------------------

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)

expandForIMS(obj, ...)

## 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'
expandForIMS(obj, fGroups)

## 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(window = 5), ...)

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

## S4 method for signature 'componentsCamera'
expandForIMS(obj, ...)

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

```

## S4 method for signature 'componentsCliqueMS'
expandForIMS(obj, ...)

## 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'
expandForIMS(obj, fGroups)

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

## S4 method for signature 'componentsNT'
expandForIMS(obj, ...)

## S4 method for signature 'componentsNTSet'
expandForIMS(obj, ...)

## S4 method for signature 'componentsOpenMS'
expandForIMS(obj, ...)

## S4 method for signature 'componentsRC'
expandForIMS(obj, ...)

```

Arguments

obj, object, x	The component object.
fGroup	The name (thus a character) of the feature group that should be searched for.
...	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 groupBy, 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.
i, j	For <code>[]</code> : A numeric or character value which is used to select components/feature groups by their index or name, respectively (for the order/names see <code>names()/groupNames()</code>). For <code>[]</code> : Can also be logical to perform logical selection (similar to regular vectors). If missing all components/feature groups are selected. For <code>[]</code> : 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 <code>data.table</code>), 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).
drop	ignored.
name	The component name (partially matched).
fGroups	The featureGroups object that was used to generate the components. For <code>expandForIMS</code> , this should be the <code>featureGroups</code> object that contains the post IMS feature groups.
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 <code>adducts</code> is a logical then only results are kept when an adduct is assigned (<code>adducts=TRUE</code>) or not assigned (<code>adducts=FALSE</code>). Otherwise, if <code>adducts</code> contains one or more adduct objects (or something that can be converted to it with <code>as.adduct</code>) 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 <code>isotopes</code> is a logical then only results are kept with (<code>isotopes=TRUE</code>) or without (<code>isotopes=FALSE</code>) isotope assignment. Otherwise <code>isotopes</code> 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 <code>checkComponentsSession</code> should either be a path to the session file or TRUE, in which case the default session file name is used. If <code>negate=TRUE</code> 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>i.e.</i> 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.
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.
- `expandForIMS(components)`: Expands the components data for IMS feature groups. See the [IMS expansion](#) section below.
- `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`

IMS expansion

In IMS workflows with post mobility assignment (see [assignMobilities](#)), specifically when the assignment occurs *after* generation of the components, it may be desired to copy the results of the IMS precursors to the IMS feature groups. For instance, for components from [intensity clusters](#) or [TPs](#) one could assume that results for IMS feature groups will be largely the same as their IMS precursors. The `expandForIMS` method function is used to *expand* the original components object by adding in IMS feature groups with data copied from their IMS precursors.

Currently, only components generated with [generateComponentsIntClust](#), [generateComponentsSpecClust](#) and [generateComponentsTPs](#) support this operation.

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 `topMostByReplicate` EIC parameters are ignored unless the components are from homologous series.

See Also

[generateComponents](#)

`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

`dism` 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.

IMS workflows

When components are re-made by `treeCut` or `treeCutDynamic` any expanded data should be re-added by calling `expandForIMS`.

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](#)

componentsIntClust-class

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](#)

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 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 (2026). *igraph: Network Analysis and Visualization in R*. doi:10.5281/zenodo.7682609, R package version 2.2.3, <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, candidates = FALSE)
```

```
## S4 method for signature 'componentsTPs'  
filter(  
  obj,
```

```

...,
retDirMatch = FALSE,
minSpecSim = NULL,
minSpecSimPrec = NULL,
minSpecSimBoth = NULL,
minTotFragMatches = NULL,
minTotNLMatches = 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.
candidates	If TRUE then candidate specific tables are merged in the result table. See generateComponentsTPs (Result columns section) for more details.
..., verbose	Further arguments passed to the base filter method .
retDirMatch	If set to TRUE, only keep TPs for which the retention order direction from feature group data matches that of the expected values from TP data (retDir and TP_retDir columns). TPs will never be removed if either of the directions is '0' (<i>i.e.</i> 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.
minTotFragMatches, minTotNLMatches, minFragMatches, minNLMatches	Minimum number of (total) parent/TP fragment and neutral loss matches. Set to NULL to ignore. See the Result columns 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 (*i.e.* the TPs argument was not NULL).

`parentsFromScreening` A logical that is TRUE when the parents were obtained from screening data.

`TPsFromScreening` A logical that is TRUE when the TPs were obtained from screening 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 (2026). *igraph: Network Analysis and Visualization in R*. doi:10.5281/zenodo.7682609, R package version 2.2.3, <https://CRAN.R-project.org/package=igraph>.

See Also

[components](#) for other relevant methods and [generateComponents](#)

compounds-class	<i>Compound annotations class</i>
-----------------	-----------------------------------

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,  
  IMSRangeParams = NULL,  
  IMSMatchParams = 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,
  normalized = "multiple",
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  showLegend = TRUE,
  maxMolSize = c(0.2, 0.4),
  molRes = c(100, 100),
  ...
)

## S4 method for signature 'compounds'
consensus(
  obj,
```

```
    ...,
    MSPeakLists,
    specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
    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,
  normalized = "multiple",
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  showLegend = TRUE,
  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,
  ...,
  MSPeakLists,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  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

<code>formulas</code>	The <code>formulas</code> object that should be used for scoring/annotation. For <code>plotSpectrum</code> and <code>annotatedPeakList</code> : set to <code>NULL</code> to ignore.
<code>updateScore</code> , <code>formulaScoreWeight</code>	If <code>updateScore=TRUE</code> then the annotation score column is updated by adding normalized values of the formula score (weighted by ‘ <code>formulaScoreWeight</code> ’). Currently, this only makes sense for annotations performed with <code>MetFrag</code> !
<code>obj</code> , <code>object</code> , <code>compounds</code> , <code>x</code>	The compound object.
<code>minExplainedPeaks</code> , <code>scoreLimits</code>	Passed to the <code>featureAnnotations</code> 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.
IMSRangeParams	(IMS workflow) A list with parameters to be used for filtering IMS range data. See getIMSRangeParams for details and how to make such a parameter list.
IMSMatchParams	(IMS workflow) A list with parameters to be used for matching IMS data. See getIMSMatchParams for details and how to make such a parameter list.
...	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>i.e.</i> 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 for which a plot should be made. To compare spectra, two group names can be specified.
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 (<i>e.g.</i> by use of filter).
excludeNormScores	A character vector specifying any compound scoring names that should <i>not</i> 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 with relevant spectral data.
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.
normalized	Controls intensity normalization. Should be FALSE (don't normalize), TRUE (normalize) or "multiple" (only normalizes if multiple spectra are plotted).
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.
showLegend	Set to TRUE to show a legend.
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 <code>algorithm(obj)</code>). 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 annotation 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 consensus 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 (<i>e.g.</i> 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 **rdk** 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 **rdk**.
- plotStructure(compounds): Plots a structure of a candidate compound using the **rdk** 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`.

(**sets workflow**) A named list with the metadata stored for each set.

`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.

IMS workflows

In IMS workflows, reference IMS data to candidates can be assigned with `assignMobilities` method function. Furthermore, CCS values may be assigned directly to candidates with `generateCompounds` if `database="pubchemlite"`.

This data can be used to prioritize candidates with the `IMSMatchParams` and `IMSRangeParams` filters.

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 `updateConsensus`, 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`.

compounds-cluster	<i>Hierarchical clustering of compounds</i>
-------------------	---

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", ...)\n\n## S4 method for signature 'compounds'\nmakeHCluster(\n  obj,\n  method,\n  fpType = "extended",\n  fpSimMethod = "tanimoto",\n  maxTreeHeight = 1,\n  deepSplit = TRUE,\n  minModuleSize = 1\n)
```

Arguments

<code>obj</code>	The <code>compounds</code> object to be clustered.
<code>method</code>	The clustering method passed to <code>hclust</code> .
<code>...</code>	further arguments specified to methods.

fpType	The type of structural fingerprint that should be calculated. See the type argument of the <code>get.fingerprint</code> function of rdck .
fpSimMethod	The method for calculating similarities (i.e. not dissimilarity!). See the method argument of the <code>fp.sim.matrix</code> function of the fingerprint package.
maxTreeHeight, deepSplit, minModuleSize	Arguments used by <code>cutreeDynamicTree</code> .

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 **rdck**.

References

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

See Also

[compoundsCluster](#)

`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 <code>groupNames()</code>). 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 (<code>plot</code> or <code>plot.dendrogram</code>).
drop, j	ignored.
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 <code>cutree</code> .
groupName	A character specifying the feature group name.
maxTreeHeight, deepSplit, minModuleSize	Arguments used by <code>cutreeDynamicTree</code> .
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 <code>plot</code> .

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 **rdk** molecule object (IAAtomContainer).

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 **rdk**.
- 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	<i>Scorings terms for compound candidates</i>
------------------	---

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 (<i>e.g.</i> "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`

compoundsMF-class	<i>Compounds list class for MetFrag results.</i>
-------------------	--

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](#)

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`

`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.

EIXParams

*Extracted Ion Chromatogram and Mobilogram parameters***Description**

Parameters for creation of extracted ion chromatograms and mobilograms.

Usage

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

```
getDefEIMParams(..., IMS = getLimIMS())
```

Arguments

...	optional named arguments that override defaults.
IMS	A character that specifies for which IMS instrument defaults are returned. Should be "bruker" or "agilent". Defaults to what is specified in limits .

Details

The following parameters exist to configure the creation of extracted ion chromatograms (EICs) and extracted ion mobilograms (EIMs):

- **window** A value that is subtracted or added to the minimum and maximum retention time (EICs) or mobility (EIMs) of the feature. Thus, setting this value to '>0' will 'zoom out' on the x-axis of a chromatogram or mobilogram. Defaults to `defaultLim("retention", "wide")` (EICs) or `defaultLim("mobility", "wide")` (EIMs) (see [limits](#)).
- **topMost** Only create EICs/EIMs for this number of top most intense features. If NULL then EICs/EIMs are created for all features.
- **topMostByReplicate** If set to TRUE and topMost is set: only create EICs/EIMs for the top most features in each replicate. For instance, when topMost=1 and topMostByReplicate=TRUE, then only the most intense feature of each replicate is considered.
- **onlyPresent** If TRUE then EICs/EIMs are created only for analyses in which a feature was detected, if onlyPresent=FALSE then data is generated for **all** analyses. The latter is handy to evaluate if a peak was 'missed' during peak detection or removed during *e.g.* filtering.
- **mzExpMobWindow (IMS workflow)** Additional *m/z* tolerance on top of the feature limits. This is for IMS workflows where features were detected from centroided LC-MS like data, while EICs/EIMs are generated from raw IMS data. In this case the feature *m/z* limits were derived from centroided data, which typically has smaller *m/z* deviations across scans compared to IMS data. The `mzExpMobWindow` parameter sets an additional *m/z* tolerance to specifically handle this case. Defaults to `defaultLim("mz", "default")` (see [limits](#)).
- **minIntensityIMS (IMS workflow)** Raw intensity threshold for IMS data. This is primarily intended to speed up raw data processing.

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

- `mzExpWindow`, `mobExpWindow` To create EICs or EIMs for analyses in which no feature was found, the m/z or mobility value is derived from the min/max values of all features in the feature group. The value of `mzExpWindow` and `mobExpWindow` further expands this window to allow a greater tolerance. Defaults to `defaultLim("mz", "very_narrow")` and `defaultLim("mobility", "very_narrow")` (see [limits](#)).
- `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.

The following additional parameters exist specifically for EICs (EICParams):

- `gapFactor` Bruker TIMS data (and maybe others?) seem to omit zero intensity scans, which will lead to time gaps between spectra and incorrect EICs. To determine a time gap, the `gapFactor` is multiplied with the median of time differences between scans. If a gap is detected, then appropriate zero intensity points are added to the EIC. Set to 0 to disable this.

The following additional parameters exist specifically for EIMs (EIMParams):

- `maxRTWindow` Maximum retention time window (seconds, +/- feature retention time) in which mobilograms are collected and averaged. Defaults to `defaultLim("retention", "very_narrow")` (see [limits](#)).
- `smooth` The smoothing method that is applied to the EIM. Can be "none" for no smoothing, "sg" for Savitzky-Golay (using [signal](#)) or "ma" for centered moving average (same algorithm as used by [findFeaturesPiek](#)).
- `smLength` The smoothing length. If `smooth="sg"` then this is passed as the `n` argument to the `signal::sgolayfilt` function.
- `sgOrder` The smoothing order for Savitzky-Golay. Passed as the `p` argument to the `signal::sgolayfilt` function.

These parameters are passed as a named list as the `EICParams` or `EIMParams` argument to functions that work with EIC or EIM data. The `getDefEICParams` and `getDefEIMParams` functions generate such parameter list with defaults.

feature-filtering	<i>Filtering of grouped features</i>
-------------------	--------------------------------------

Description

Basic rule based filtering of feature groups.

Usage

```
replicateSubtract(fGroups, replicates, threshold = 0)
```

```
## S4 method for signature 'featureGroups'
filter(
```



```
obj,
absMinIntensity = NULL,
relMinIntensity = NULL,
preAbsMinIntensity = NULL,
preRelMinIntensity = NULL,
absMinMaxIntensity = NULL,
relMinMaxIntensity = 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,
replicates = NULL,
IMS = NULL,
withIMSPrecursor = FALSE,
IMSRangeParams = NULL,
results = NULL,
removeBlanks = FALSE,
removeISTDs = FALSE,
checkFeaturesSession = NULL,
predAggrParams = getDefPredAggrParams(),
removeNA = FALSE,
negate = FALSE,
applyIMS = "both"
)

## S4 method for signature 'featureGroupsSet'
filter(
  obj,
  ...,
  negate = FALSE,
```

```

    applyIMS = "both",
    sets = NULL,
    absMinSets = NULL,
    relMinSets = NULL
)

## S4 method for signature 'featureGroups'
replicateSubtract(fGroups, replicates, threshold = 0)

```

Arguments

- fGroups, obj** **featureGroups** object to which the filter is applied.
- replicates** A character vector of replicates that should be kept (filter) or subtracted from (replicateSubtract).
- threshold** Minimum relative threshold (compared to mean intensity of replicate being subtracted) for a feature group to be *not* removed. When '0' a feature group is always removed when present in the given replicates.
- 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.
- absMinMaxIntensity, relMinMaxIntensity** Feature groups are only kept if at least one feature in the group has an intensity above this absolute/relative threshold.
- 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. If this minimum is not met all features within the replicate 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. If the RSD is above this value all features within the replicate 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 [getFeatureQualityNames](#) 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.

IMS

(**IMS workflow**) Specifies which feature groups are considered to be kept in IMS workflows. The following options are valid:

- "both": Selects IMS and non-IMS features.
- "maybe": Selects non-IMS features and IMS features without assigned IMS precursor.
- FALSE: Selects only non-IMS features.
- TRUE: Selects only IMS features.

Set to NULL to ignore.

withIMSPrecursor

(**IMS workflow**) only keep IMS feature groups with IMS precursors, *i.e.* remove all orphans. Unaffected by `negate=TRUE`.

IMSRangeParams	(IMS workflow) A list with parameters to be used for filtering IMS range data. See getIMSRangeParams for details and how to make such a parameter list.
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 <i>any</i> of the objects. Set to NULL to ignore.
removeBlanks	Set to TRUE to remove all analyses that belong to replicates 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 be 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/calculateToxicities) 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.
applyIMS	(IMS workflow) whether the filters are only applied to IMS precursors (applyIMS=FALSE), only to IMS features (applyIMS=TRUE) or to both (applyIMS="both"). Other feature groups will always be kept. The negate option does not affect applyIMS.
...	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.

replicateSubtract removes feature groups present in a given set of replicates (unless intensities are above a given threshold). The replicates 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.
- **Important:** the `mzRange`, `mzDefectRange` and `IMSRangeParams` filters use neutral feature masses, whereas non-sets workflows use m/z values. Hence, adjust accordingly to avoid (slightly) different results!

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.

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. Minimum-maximum intensity filters (*i.e.* `absMinMaxIntensity` and `relMinMaxIntensity`).
9. General abundance filters (*i.e.* `absMinAnalyses`, `relMinAnalyses`, `absMinReplicates`, `relMinReplicates`, `absMinFeatures`, `relMinFeatures`, `absMinConc`, `relMinConc`, `absMaxTox` and `relMaxTox`).
10. Replicate filter (*i.e.* `replicates`), 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](#)

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 chosen 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 optimum values. Used to restrict needless 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 future 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 patRoön. 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 parameters 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.

Similarly, 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 optimum 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 significantly lower than expected. When the response is lower than the maximum response found during the experiment, the parameters belonging to this experimental maximum may be chosen 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 patRoonaData package
dataDir <- patRoonaData::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

```
plotMobilograms(obj, ...)
```

```
## S4 method for signature 'featureGroups,missing'
```

```
plot(
  x,
```

```

    groupBy = NULL,
    onlyUnique = FALSE,
    retMin = FALSE,
    showLegend = TRUE,
    IMS = "maybe",
    col = NULL,
    pch = NULL,
    ...
)

## S4 method for signature 'featureGroups'
plotInt(
  obj,
  average = FALSE,
  averageFunc = mean,
  areas = FALSE,
  normalized = FALSE,
  xBy = NULL,
  xNames = TRUE,
  groupBy = "fGroups",
  regression = FALSE,
  showLegend = FALSE,
  IMS = "maybe",
  pch = 20,
  type = if (regression) "p" else "b",
  lty = 3,
  xlim = NULL,
  ylim = NULL,
  col = NULL,
  plotArgs = NULL,
  linesArgs = NULL
)

## S4 method for signature 'featureGroups'
plotChord(
  obj,
  addSelfLinks = FALSE,
  addRetMzPlots = TRUE,
  aggregate = FALSE,
  groupBy = NULL,
  addIntraOuterGroupLinks = FALSE,
  IMS = "maybe",
  ...
)

## S4 method for signature 'featureGroups'
plotChroms(
  obj,

```

```

    analysis = analyses(obj),
    groupName = names(obj),
    retMin = FALSE,
    showPeakArea = FALSE,
    showFGroupRect = TRUE,
    title = NULL,
    groupBy = NULL,
    showLegend = TRUE,
    annotate = c("none", "ret", "mz", "mob"),
    intMax = "eic",
    EICParams = getDefEICParams(),
    showProgress = FALSE,
    IMS = "maybe",
    xlim = NULL,
    ylim = NULL,
    EICs = NULL,
    ...
)

## S4 method for signature 'featureGroups'
plotChroms3D(
  obj,
  analysis = analyses(obj),
  groupName = names(obj),
  dim3 = "mz",
  retMin = FALSE,
  showLimits = TRUE,
  rtWindow = defaultLim("retention", "medium"),
  mzWindow = defaultLim("mz", "medium"),
  mobWindow = defaultLim("mobility", "medium"),
  gridSize = 50,
  title = NULL,
  ...
)

## S4 method for signature 'featureGroupsSet'
plotChroms3D(obj, analysis = analyses(obj), ...)

## S4 method for signature 'featureGroups'
plotMobilograms(
  obj,
  analysis = analyses(obj),
  groupName = names(obj),
  showPeakArea = FALSE,
  showFGroupRect = TRUE,
  title = NULL,
  groupBy = NULL,
  showLegend = TRUE,

```

```

        annotate = c("none", "ret", "mz", "mob"),
        EIMParams = getDefEIMParams(),
        showProgress = FALSE,
        xlim = NULL,
        ylim = NULL,
        EIMs = NULL,
        ...
    )

## S4 method for signature 'featureGroups'
plotVenn(obj, which = NULL, aggregate = TRUE, IMS = "maybe", ...)

## S4 method for signature 'featureGroups'
plotUpSet(
    obj,
    which = NULL,
    aggregate = TRUE,
    IMS = "maybe",
    nsets = NULL,
    nintersects = NA,
    ...
)

## S4 method for signature 'featureGroups'
plotVolcano(
    obj,
    FCParams,
    showLegend = TRUE,
    averageFunc = mean,
    normalized = FALSE,
    IMS = "maybe",
    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, ...)

## S4 method for signature 'featureGroups'
plotTICs(
    obj,
    retentionRange = NULL,
    MSLevel = 1,
    retMin = FALSE,

```

```

    title = NULL,
    groupBy = NULL,
    showLegend = TRUE,
    xlim = NULL,
    ylim = NULL,
    ...
)

## S4 method for signature 'featureGroups'
plotBPCs(
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  groupBy = NULL,
  showLegend = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)

```

Arguments

obj, x	featureGroups object to be used for plotting.
...	passed to <code>plot</code> (plot, plotChroms, plotMobilograms, plotTICs and plotBPCs), <code>filled.contour</code> (plotChroms3D), VennDiagram plotting functions (plotVenn), <code>chordDiagram</code> (plotChord), <code>upset</code> (plotUpSet).
groupBy	Specifies how results are grouped in the plot. Should be a name of a column in the <code>analysis information</code> table which is used to make analysis groups (<i>e.g.</i> "replicate"). Set to NULL for no grouping. The groupBy argument can also be set to "fGroups" to group by feature groups (except plotChord). For plotInt: see the Data aggregation in intensity plots section for more details. For plotChord: the grouping is used to generate 'outer groups'.
onlyUnique	If TRUE and groupBy is set to a column name of the <code>analysis information</code> then only feature groups that are unique to a replicate are plotted.
retMin	Plot retention time in minutes (instead of seconds).
showLegend	Plot a legend if TRUE.
IMS	(IMS workflow) Specifies which feature groups are considered for plotting in IMS workflows. The following options are valid: <ul style="list-style-type: none"> • "both": Selects IMS and non-IMS features. • "maybe": Selects non-IMS features and IMS features without assigned IMS precursor. • FALSE: Selects only non-IMS features. • TRUE: Selects only IMS features.

	For plotChroms: if the groupName argument is set to a subset of the feature groups, then the IMS argument is forced to "both" to ensure that the selected feature groups are plotted.
col	Colour(s) used. If col=NULL then colours are automatically generated.
pch, type, lty	Common plotting parameters passed to <i>e.g.</i> <code>plot</code> . For plot: if pch=NULL then values are automatically assigned. for plotInt: the type argument defaults to single points ("p") when regression=TRUE, and lines+points ("b") otherwise. For non-aggregated data, it probably makes more sense to set <i>e.g.</i> type="p" to avoid lines being drawn to points with equal x-values.
average	Controls plot data averaging: see the Data aggregation in intensity plots section.
averageFunc, normalized	Used for intensity data treatment, see the documentation for the as.data.table method .
areas	Set to TRUE to use feature areas instead of peak intensities for plotting.
xBy	Controls x-value grouping in the plot: see the Data aggregation in intensity plots section.
xNames	Plot names (xNames=TRUE) or a number sequence (xNames=FALSE) on the x axis.
regression	If TRUE then a regression line is plotted for each group, using the x values set by xBy (see Data aggregation in intensity plots). if showLegend=TRUE then the R-squared value is show in the legend for each group (unless there are multiple feature groups and groupBy != "fGroups").
xlim, ylim	Sets the plot size limits used by <code>plot</code> . Set to NULL for automatic plot sizing.
plotArgs, linesArgs	A list with further arguments passed to <code>plot</code> and <code>lines</code> , respectively.
addSelfLinks	If TRUE then 'self-links' are added which represent non-shared data.
addRetMzPlots	Set to TRUE to enable <i>m/z</i> vs retention time scatter plots.
aggregate	Specifies how data should be aggregated prior to comparison. Set to FALSE to compare analyses, TRUE to compare replicates or to the name of a column in the analysis information to compare by a custom grouping of analyses.
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. For plotChroms and plotMobilograms: Compared to subsetting the featureGroups object (obj) upfront this is slightly faster. Furthermore, if onlyPresent=FALSE in EICParams or EIMParams, this 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). For IMS workflows: if IMS!="both" then the analysis and groupName arguments are adjusted for the remaining data after IMS selection. For plotChroms3D: a single analysis and group name should be specified (unless obj contains only a single feature, <i>e.g.</i> created as fGroups[1, 1]).

showPeakArea	Set to TRUE to display integrated peak ranges by filling (shading) their areas.
showFGroupRect	Set to TRUE to mark the full integration/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	Set to "ret", "mz" and/or "mob" to annotate peaks with the retention time, m/z and/or ion mobility (if available) feature group values, respectively.
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.
EICs, EIMs	Internal parameter for now and should be kept at NULL (default).
dim3	The third dimension to plot besides retention time and intensity. Can be either "mz" or "mobility".
showLimits	If TRUE, a rectangle is drawn to indicate the feature limits.
rtWindow, mzWindow, mobWindow	Numeric values specifying the window size around the feature for retention time, m/z , and ion mobility respectively. Values >0 will effectively zoom out.
gridSize	The size of the grid for interpolation.
EIMParams	A named list with parameters used for extracted ion mobilogram (EIM) creation. See the EIM parameters documentation for more details.
which	A character vector with the selection to compare (<i>e.g.</i> replicates, as set by the aggregate argument). Set to NULL to select everything.
nsets, nintersects	See upset . If nsets == NULL then it will be set to the number of compared items.
FCParams	A parameter list to calculate Fold change data. See getFCParams for more details.
onlyPresent	Only plot feature groups of internal standards that are still present in the featureGroups input object (which may be otherwise be removed by <i>e.g.</i> subsetting or filter).
width, height	Passed to visNetwork .
set	(sets workflow) The set for which data must be plotted.
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>i.e.</i> , 1 for MS1 and 2 for MS2) to obtain TIC traces.

Details

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

`plotInt` Generates a line plot with feature intensities.

`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.

`plotChroms3D` generates a 3D plot of chromatographic data for a single feature group in a single analysis. The plot shows retention time on the x-axis, m/z or mobility on the y-axis, and intensity as color in a contour plot. The plot is made with the `filled.contour` function.

`plotMobilograms` Plots extracted ion mobilograms (EIMs) of feature groups.

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

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

`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**.

`plotTICs` Plots the total ion chromatogram/s (TICs) of the analyses.

`plotBPCs` Plots the base peak chromatogram/s (BPCs) of the analyses.

Value

`plotChroms3D` returns the interpolated grid used for plotting (generated with `mba.surf`).

`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**.

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by `plotChroms`, `plotMobilograms`, `plotTICs` and `plotBPCs` to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

Sets workflows

The following methods are changed or with new functionality:

- `plotGraph` only plots data per set, and requires the `set` argument to be set.

In sets workflows the [analysis information](#) contains an additional "set" column, which can be used for arguments that involve grouping of analyses. For instance, if `groupBy="set"` then plotting data is grouped per set.

Data aggregation in intensity plots

the average, `xBy` and `groupBy` arguments control how data is aggregated in intensity plots:

- `average`: controls the averaging of feature intensities prior to plotting.
- `xBy`: can map the x value of individual points to analysis metadata. For example, exposure time or sample location. Non-numeric values are allowed (unless `regression=TRUE`).
- `groupBy`: controls the grouping of points in the plot. Equal groups are plotted in sequence so they can be connected with lines and are coloured equally. Examples include experiment type or feature groups.

The following values are valid:

- `FALSE` (average) or `NULL` (`xBy` and `groupBy`): aggregation is disabled.
- `TRUE` (only average): results are averaged for each replicate
- a name of a column in the [analysis information](#): results are aggregated for analyses with the same table column value.
- "fGroups" (only `groupBy`): plots are grouped by feature groups.

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, Strobel H, Vuilleumot 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](#)

Description

Functions to define and work with chromatographic peak quality metrics used for feature and feature group quality calculations.

Usage

```
featureQualities(qualities = NULL)
```

```
featureGroupQualities(qualities = NULL)
```

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

Arguments

qualities	A character vector specifying which qualities to return. If NULL, returns all available quality definitions.
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.

Details

These functions provide access to quality metrics that are used to assess the quality of detected features and feature groups by the [calculatePeakQualities](#) function. The quality metrics are calculated using the **MetaClean** package and are useful for filtering out low-quality features before further analysis.

Value

For `featureQualities` and `featureGroupQualities`: A named list containing quality definitions. Each element contains:

- `func`: The MetaClean function to calculate the quality metric
- `HQ`: "HV" (high values good) or "LV" (low values good)
- `range`: Expected range of values (may be Inf for unbounded ranges)

For `featureQualityNames`: A character vector with quality and/or score names.

Feature qualities

The `featureQualities` function defines quality metrics that are calculated for individual features. The following quality metrics are available:

- `ApexBoundaryRatio` - Ratio between apex and boundary intensities
- `FWHM2Base` - Full width at half maximum to base ratio
- `Jaggedness` - Measure of peak smoothness
- `Modality` - Measure of peak multiplicity
- `Symmetry` - Measure of peak symmetry (range: -1 to 1)
- `GaussianSimilarity` - Similarity to Gaussian distribution
- `Sharpness` - Measure of peak sharpness
- `TPASR` - Triangle peak area similarity ratio
- `ZigZag` - Zig-zag index measure

Feature group qualities

The `featureGroupQualities` function defines quality metrics that are calculated for feature groups across multiple analyses. The following quality metrics are available:

- `ElutionShift` - Measure of retention time consistency across analyses
- `RetentionTimeCorrelation` - Correlation of retention times across analyses

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

[calculatePeakQualities](#)

feature-table

Feature group to table conversion

Description

Convert feature group data to a `data.table` (or `data.frame`).

Usage

```
## S4 method for signature 'featureGroups'
as.data.table(
  x,
  average = FALSE,
  areas = FALSE,
  features = FALSE,
  qualities = FALSE,
  regression = FALSE,
  regressionBy = NULL,
  averageFunc = mean,
  normalized = FALSE,
  FCParams = NULL,
  concAggrParams = getDefPredAggrParams(),
  toxAggrParams = getDefPredAggrParams(),
  normConcToTox = FALSE,
  anaInfoCols = NULL,
  IMS = "both"
)

## S4 method for signature 'featureGroupsScreening'
as.data.table(x, ..., collapseSuspects = ",", onlyHits = FALSE)

## S4 method for signature 'featureGroupsScreeningSet'
as.data.table(x, ..., collapseSuspects = ",", onlyHits = FALSE)
```

Arguments

x	The featureGroups like object to be exported as table.
average	Controls the averaging of feature intensities. Averaging also influences the calculation of regression parameters. Set to FALSE to disable averaging, TRUE to average per replicate or to the name of a column in the analysis information to compare by a custom grouping of analyses. If features=TRUE all numerical properties are averaged and non-numericals are collapsed. Each row represents the feature aggregated data and the groups used for aggregation (<i>e.g.</i> replicates) are stored in the average_group column. It is also possible to average these data per feature group, by setting average="fGroups".
areas	If set to TRUE then areas are output instead of peak intensities. Ignored if features=TRUE, as areas of features are always reported.
features	If TRUE then feature specific data will be added. Also see the average argument.
qualities	Adds feature (group) qualities (qualities="quality"), scores (qualities="score") or both (qualities="both"), if this data is available (<i>i.e.</i> from calculatePeakQualities). If qualities=FALSE then nothing is reported.
regression, regressionBy	Used for regression calculations. See the Regression calculation section below. Set to NULL to ignore.

averageFunc	Function used for averaging. Only used when data is averaged or FCParams != NULL.
normalized	If TRUE then normalized intensities are used (see the Feature intensity normalization section). If no normalization data is available (<i>e.g.</i> because normInts was not used) then an automatic group normalization is performed.
FCParams	A parameter list to calculate fold change data. See getFCParams for more details. Set to NULL to not perform FC calculations. NOTE: the intensity data is always averaged to calculate fold changes (using averageFunc) by replicates, irrespective of the average argument.
concAggrParams, toxAggrParams	Parameters to aggregate calculated concentrations/toxicities (obtained with calculateConcs/calculateTox). 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).
anaInfoCols	A character with any additional columns from the analysis information table. Only supported if features=TRUE. If averaging is performed then the data in the specified columns should be numeric. Set to NULL to ignore.
IMS	(IMS workflow) Specifies which feature groups are considered to be returned in IMS workflows. The following options are valid: <ul style="list-style-type: none"> • "both": Selects IMS and non-IMS features. • "maybe": Selects non-IMS features and IMS features without assigned IMS precursor. • FALSE: Selects only non-IMS features. • TRUE: Selects only IMS features.
...	Passed to the parent as <code>.data.table</code> method.
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	If TRUE then only feature groups with suspect hits are reported.

Details

The `as.data.table` generic function converts most feature group data to a highly customizable [data.table](#). If a classical `data.frame` is preferred, the `as.data.frame` generic function can be used instead and accepts the exact same arguments. The methods defined for [suspect screening workflows](#) will merge the information from `screenInfo`, such as suspect names and other properties and annotation data.

Regression calculation

The `regression` argument controls the calculation of regression parameters from a regression model calculated with feature intensities (or areas if `areas=TRUE`). Here, simple linear regression

is used, *i.e.* 'y=ax+b' with 'a' the slope and 'b' the intercept. The value for regression should be the name of a column in the [analysis information](#) table with numerical data to be used for x-values. Alternatively, if regression=TRUE then the "conc" column is used. Any NA x-values are ignored, and no regression will be calculated if less than two (non-NA) x-values are available. The output table will contain properties such as the slope and correlation coefficient (R-squared). Furthermore, if features=TRUE then x-values will be calculated from the model and stored in the x_reg column.

The regressionBy argument can be used to construct separate regression models for different groups of analysis. It should be set to the name of a column in the [analysis information](#) table which defines the grouping between samples. If features=TRUE then the grouping is stored in the regression_group column of the output table.

Please see the handbook for examples on how to use the regression functionality.

Suspect collapsing

The as.data.table method for 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 and estimated ID levels (if available) of the suspects matched to a feature group (each separated by the value of the collapseSuspects argument).

Suspect collapsing also influences the reporting of predicted [feature concentrations](#) and [toxicities](#). In the case that (1) suspects are *not* collapsed in the output table and (2) predictions are available for a specific suspect hit (*i.e.* if [predictRespFactors](#) or [predictTox](#) was called on the feature groups object), then only the suspect specific data is reported and no aggregation is performed. Hence, this allows you to obtain specific concentration/toxicity values for each suspect/feature group pair.

IMS workflows

If the IMS argument is set to "both" or "maybe" then "mobility_collapsed" and "CCS_collapsed" columns will be added that summarize all mobility/CCS values of the IMS features (or IMS feature groups) assigned to this IMS precursor. These numbers are currently rounded to '3' decimals.

Sets workflows

In a [sets workflow](#) normalization of feature intensities occur per set.

In sets workflows the [analysis information](#) contains an additional "set" column, which can be used for arguments that involve grouping of analyses. For instance, if regressionBy="set" then regression models will be calculated for each set.

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,
  fragFormulas = NULL,
  lossFormulas = NULL,
  topMost = NULL,
  OM = FALSE,
  maxLevel = NULL,
  negate = FALSE
```



```

)

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

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

```

Arguments

obj, x	featureAnnotations object to be accessed
i, j	For <code>[]</code> : A numeric or character value which is used to select feature groups by their index or name, respectively (for the order/names see <code>groupNames()</code>). For <code>[]</code> : Can also be logical to perform logical selection (similar to regular vectors). If missing all feature groups are selected. For <code>[]</code> : should be a scalar value. For delete: The data to remove from. <code>i</code> are the feature groups as numeric index, logical or character, <code>j</code> the candidates as numeric indices (rows). If either is <code>NULL</code> then data for all is removed. <code>j</code> may also be a function: it will be called for each feature group, with the annotation table (a <code>data.table</code>), the feature group name and any other arguments passed as <code>...</code> 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 <code>"["</code> operator: ignored. For delete: passed to the function specified as <code>j</code> . 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 <code>NULL</code> it is used to add feature group information (retention and <i>m/z</i> values).
fragments	If <code>TRUE</code> then information on annotated fragments will be included. Automatically set to <code>TRUE</code> if <code>countFragElements</code> is set.
countElements, countFragElements	A character vector with elements that should be counted for each candidate's formula. For instance, <code>c("C", "H")</code> adds columns for both carbon and hydrogen amounts of each formula. Note that the neutral formula (<code>neutral_formula</code>

	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	<p>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.</p> <p>For filter: If TRUE then several filters are applied to exclude unlikely formula candidates present in organic matter (OM). See Source section for details.</p>
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	<p>A character vector specifying any compound scoring names that should <i>not</i> be normalized. Set to NULL to normalize all scorings. Note that whether any normalization occurs is set by the excludeNormScores argument.</p> <p>For compounds: By default score and individualMoNAScore are set to mimic the behavior of the MetFrag web interface.</p>
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>i.e.</i> 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 formulae 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 <code>negate=TRUE</code>) the given restrictions. See <code>elements</code> for the used format.
<code>fragFormulas, lossFormulas</code>	A character vector with one or more formulae, which are matched to MS/MS fragments (<code>fragFormulas</code>) or neutral losses (<code>lossFormulas</code>). Candidates are only kept with at least one match (if both <code>fragFormulas</code> and <code>lossFormulas</code> are set then there must be at least one match from both). The input formulae are hill sorted prior to matching. For non-exact matching: see <code>fragElements</code> and <code>lossElements</code> . Set to <code>NULL</code> to ignore these filters.
<code>topMost</code>	Only keep a maximum of <code>topMost</code> candidates with highest score (or least highest if <code>negate=TRUE</code>). Set to <code>NULL</code> to ignore.
<code>maxLevel</code>	Filter by maximum identification level (e.g. "3a"). Set to <code>NULL</code> to ignore.
<code>negate</code>	If <code>TRUE</code> then filters are applied in opposite manner.
<code>labels</code>	A character with names to use for labelling. If <code>NULL</code> labels are automatically generated.
<code>vennArgs</code>	A list with further arguments passed to VennDiagram plotting functions. Set to <code>NULL</code> to ignore.
<code>nsets, nintersects</code>	See upset . If <code>nsets == NULL</code> then it will be set to the number of compared items.
<code>upsetArgs</code>	A list with any further arguments to be passed to upset . Set to <code>NULL</code> 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, Strobel 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.

featureGroups-class	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 = NULL, aggregate = TRUE, 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,
  ...
)

concentrations(fGroups, ...)

toxicities(fGroups, ...)

updateGroups(fGroups, ...)

## S4 method for signature 'featureGroups'
names(x)

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

## S4 method for signature 'featureGroups'
replicates(obj)
```

```
## S4 method for signature 'featureGroups'
groupNames(obj)

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

## S4 method for signature 'featureGroups'
hasIMS(obj)

## S4 method for signature 'featureGroups'
fromIMS(obj)

## 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, df = FALSE)

## S4 replacement method for signature 'featureGroups'
analysisInfo(obj) <- value

## 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'
getFeatureQualityNames(
  obj,
  feat = TRUE,
  group = TRUE,
  scores = FALSE,
```

```
    totScore = TRUE
  )

## 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, ..., ni, replicates, IMS, results, reorder = FALSE, 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, IMS = FALSE)

## S4 method for signature 'featureGroups'
unique(
  x,
  incomparables = FALSE,
  which,
  aggregate = TRUE,
  relativeTo = NULL,
  outer = FALSE
)
```



```
## S4 method for signature 'featureGroups'
overlap(fGroups, which, aggregate, exclusive)

## S4 method for signature 'featureGroups'
calculatePeakQualities(
  obj,
  weights,
  flatnessFactor,
  featureQualities = NULL,
  featureGroupQualities = NULL,
  avgFunc = mean,
  EICParams = getDefEICParams(window = 0),
  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 = 1)

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

## S4 method for signature 'featureGroups'
updateGroups(
  fGroups,
  what = c("ret", "mz", "mobility", "CCS"),
```

```
    intWeight = FALSE
  )

## 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, ..., sets = NULL, reorder = FALSE, drop = TRUE]

## S4 method for signature 'featureGroupsSet'
export(obj, type, out, ..., set)

## 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 replacement method for signature 'featureGroupsXCMS'
analysisInfo(obj) <- value

## 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	Not used. Included for compatibility with the generic.
which	A character vector with the selection to compare (<i>e.g.</i> replicates, as set by the aggregate argument). For overlap: can also be a NULL to compare all elements.
aggregate	Specifies how data should be aggregated prior to comparison. Set to FALSE to compare analyses, TRUE to compare replicates or to the name of a column in the analysis information to compare by a custom grouping of analyses.
exclusive	If TRUE then all feature groups are removed that are not unique to the given replicates.
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, ISTDZWindow

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.

normalized If TRUE then normalized intensity data is used (see the Feature intensity normalization section).

df If TRUE then the returned value is a `data.frame`, otherwise a `data.table`.

value For `analysisInfo<-`: A `data.frame` or `data.table` with updated analysis information.

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.

feat, group If TRUE then names specific to features and feature groups are returned, respectively.

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.

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`).

ni Optional argument. An expression used for subsetting the analyses. The [analysis information](#) is first subset and the remaining rows are used to determine for which analyses the results should be kept. The unevaluated `ni` expression is used to set the `i` argument of the subset operator of `data.table`, which therefore brings the advanced subsetting capabilities of `data.table` (see the [data.table](#) documentation for more details). For instance, `fGroups[replicate == "standard"]` would subset all analyses assigned with the replicate "standard".

replicates	An optional character vector: if specified only keep results for the given replicates (equivalent to the replicates argument to filter).
IMS	<p>(IMS workflow) Specifies which feature groups are considered to be kept ("["") or exported (export) in IMS workflows. The following options are valid:</p> <ul style="list-style-type: none"> • "both": Selects IMS and non-IMS features. • "maybe": Selects non-IMS features and IMS features without assigned IMS precursor. • FALSE: Selects only non-IMS features. • TRUE: Selects only IMS features. <p>For "[": Leave unassigned to perform no IMS selection. For export: keep FALSE as the export formats currently do not support IMS data.</p>
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 <i>any</i> of the objects (equivalent to the results argument to filter).
reorder	<p>If TRUE then the order of the analyses is changed to match the order of the i argument.</p> <p>(sets workflow) If the sets argument is specified (and i is not) then the order of sets is changed instead.</p>
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.
relativeTo	A character vector of groupings that should be used for unique comparison. The groupings (<i>e.g.</i> replicates) are configured by the aggregate argument. If NULL then all groupings are used for comparison. Data specified in which are ignored.
outer	If TRUE then only feature groups are kept which do not overlap among those present in the groupings specified by 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 .
featureQualities	Specifies which feature qualities to calculate. Can be NULL (default, calculates all qualities), a character vector with names of qualities to calculate (<i>e.g.</i> , c("FWHM2Base", "Symmetry")), or a list of custom quality definitions. See the featureQualities function for more details.
featureGroupQualities	Analogous to featureQualities for feature groups metrics. See the featureGroupQualities function for more details.

avgFunc	The function used to average the peak qualities and scores for each feature group.
EICParams	A named list with parameters used for extracted ion chromatogram (EIC) creation. See the EIC parameters documentation for more details.
parallel	If set to TRUE then code is executed in parallel through the future 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). <i>Important:</i> 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.
what	A character vector specifying which group-wise values to update. Valid values are "ret" (retention time, in seconds), "mz" and "mobility". At least one must be specified. If "mobility" is specified and no IMS information is present, it will be ignored.
intWeight	If TRUE, calculate intensity-weighted means per group; otherwise use simple arithmetic means.
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) A character with name(s) of the sets to keep.

Details

The featureGroup class is the workhorse of **patRoön**: 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.
- `replicates(featureGroups)`: returns a character vector with the names of the replicates 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.
- `hasIMS(featureGroups)`: Returns TRUE if the feature groups object has ion mobility information.
- `fromIMS(featureGroups)`: Returns TRUE if the features were directly generated from IMS data.
- `show(featureGroups)`: Shows summary information for this object.
- `groupTable(featureGroups)`: Accessor for groups slot.
- `analysisInfo(featureGroups)`: Obtain analysisInfo (see analysisInfo slot in [features](#)).
- `analysisInfo(featureGroups) <- value`: Modifies the [analysis information](#) of this features object. This is primarily intended to change or add analysis metadata columns or can be used to re-order analysis. The removal or addition of analyses and changes to the "analysis" column are not supported. This function performs several internal updates after analysis information modifications. Hence, never attempt to change the analysisInfo slot directly.
- `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.
- `getFeatureQualityNames(featureGroups)`: Returns feature quality names that were calculated for this object.
- `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**.
- `unique(featureGroups)`: Obtain a subset with unique feature groups present in one or more analyses, replicates etc.
- `overlap(featureGroups)`: Obtain a subset with feature groups that overlap between a set of specified replicate(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. Group metrics are then calculated and scored and scaled 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 normalize 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.
- `updateGroups(featureGroups)`: Recalculate group information from feature data.

Slots

groups Matrix ([data.table](#)) with intensities for each feature group (columns) per analysis (rows). Access with `groups` method.

features [features](#) class associated with this object. Access with `featureTable` methods.

groupInfo `data.table` with retention time (`ret` column, in seconds) and *m/z* (`mz` 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 [concentrations/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](#).

Use of raw HRMS data

The [raw data interface](#) of **patRoan** is used by [calculatePeakQualities](#) to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

S4 class hierarchy

- [workflowStep](#)
 - [featureGroups](#)
 - * [featureGroupsSet](#)
 - [featureGroupsScreeningSet](#)
 - * [featureGroupsUnset](#)
 - * [featureGroupsScreening](#)
 - [featureGroupsSetScreeningUnset](#)
 - * [featureGroupsBruker](#)
 - * [featureGroupsConsensus](#)
 - * [featureGroupsEnviMass](#)
 - * [featureGroupsGreedy](#)
 - * [featureGroupsIMS](#)
 - * [featureGroupsKPIC2](#)
 - * [featureGroupsOpenMS](#)
 - * [featureGroupsSIRIUS](#)

```

* featureGroupsTable
* 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 **rdk** format, e.g. "[13]C" for 13C, "[2]H" for a deuterium etc. Example IS lists are provided with the **patRoonaData** and **patRoonaDataIMS** packages.

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 **patRoam** '<2.1', and still occurs automatically if normInts was not called when a method is executed that requests normalized data.

In IMS workflows with post mobility assignment (see [assignMobilities](#)), any IMS features are excluded for the assignment of internal standards (featNorm="istd") or calculation of TICs (featNorm="tic"). Furthermore, the normalized intensities and areas for IMS features are copied from their IMS precursors.

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.
- 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](#), [feature-table](#) and [feature-plotting](#) for more advanced featureGroups methods.

featureGroups-compare *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'
hasIMS(obj)

## 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(rtaalign = FALSE))

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

Arguments

...	<p>For comparison: featureGroups objects that should be compared. If the arguments are named (<i>e.g.</i> myGroups = fGroups) then these are used for labelling, otherwise objects are automatically labelled by their algorithm.</p> <p>For plot, plotVenn, plotChord: further options passed to plot, VennDiagram plotting functions (<i>e.g.</i> draw.pairwise.venn) and chordDiagram respectively.</p> <p>For plotUpSet: any further arguments passed to the plotUpSet method defined for featureGroups.</p>
groupAlgo	<p>The feature grouping algorithm that should be used for grouping <i>pseudo</i> features (see details). Valid values are: "xcms", xcms3, kplic2, "openms" or "greedy".</p> <p>(IMS workflow) For IMS workflows the algorithm selection is equally limited to as what is used by feature grouping, <i>i.e.</i> only greedy is currently supported.</p>
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	<p>A character vector specifying one or more labels of compared feature groups.</p> <p>For plotVenn: if NULL then all compared groups are used.</p>
addSelfLinks	If TRUE then 'self-links' are added which represent non-shared data.
addRetMzPlots	Set to TRUE to enable <i>m/z</i> vs retention time scatter plots.
absMinAbundance, relMinAbundance	<p>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.</p>
uniqueFrom	<p>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 <code>algorithm(obj)</code>). For logical and numeric values the order corresponds to the order of the objects given for the consensus. Set to NULL to ignore.</p>
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 (<i>e.g.</i> 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.

`hasIMS` returns TRUE if the object has ion mobility information.

`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.

`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.

featureGroupsScreening-class

Class for suspect screened feature groups.

Description

This class derives from [featureGroups](#) and adds suspect screening information.

Usage

```
screenInfo(obj)
```

```
## 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, ..., suspects = NULL, reorder = FALSE, drop = TRUE]
```

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

```
## S4 method for signature 'featureGroupsScreening'
filter(
  obj,
  ...,
  onlyHits = NULL,
  IMSSMatchParams = 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,
    applyIMS = "both"
)

## 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, ..., suspects = NULL, sets = NULL, reorder = FALSE, drop = TRUE]

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

## S4 method for signature 'featureGroupsScreeningSet'
filter(
  obj,
  ...,
  onlyHits = NULL,
  IMSMatchParams = 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,
  applyIMS = "both"
)

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

```

Arguments

obj, object, x	The featureGroupsScreening object.
i, j, reorder	See featureGroups .
...	Further arguments passed to the base method.
suspects	An optional character vector with suspect names. If specified, only featureGroups will be kept that are assigned to these suspects.

drop	Ignored.
k	<p>The k argument is used to delete screening results (instead of features) and should be:</p> <ul style="list-style-type: none"> • a character vector with suspect names that should be removed • a function that is called with the screening info table and should return a logical vector for each suspect row to be removed • NA to remove all screening results, which is especially useful when paired with the j argument, <i>i.e.</i> to remove all screening results for a particular set of feature groups. • NULL to not touch screening results and only perform deletion as the featureGroups method. <p>Setting both i and k is currently not supported.</p>
onlyHits	<p>If</p> <ul style="list-style-type: none"> • negate=FALSE and onlyHits=TRUE then all feature groups without suspect hits will be removed. Otherwise nothing will be done. • 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.
IMSMatchParams	(IMS workflow) A list with parameters to be used for matching IMS data. See getIMSMatchParams for details and how to make such a parameter list.
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 (<i>e.g.</i> "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>i.e.</i> 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.
applyIMS	(IMS workflow) whether the filters are only applied to IMS precursors (applyIMS=FALSE), only to IMS features (applyIMS=TRUE) or to both (applyIMS="both"). Other feature groups will always be kept. The negate option does not affect applyIMS.
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

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

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.
- delete(featureGroupsScreening): Completely deletes specified feature groups or screening results.
- 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 these filter steps.

Slots

screenInfo A ([data.table](#)) with results from suspect screening. This table will be amended with ID confidence data when [estimateIDConfidence](#) is run.

MS2QuantMeta Metadata from **MS2Quant** filled in by predictRespFactors.
(**sets workflow**) A named list with the metadata stored for each set.

S4 class hierarchy

- [featureGroups](#)
 - [featureGroupsScreening](#)
 - * [featureGroupsSetScreeningUnset](#)

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:

- `estimateIDConfidence` See the Sets workflows section in the documentation for [estimateIDConfidence](#).
- `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`).

See Also

[featureGroups](#)

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, df = FALSE)
```

```
## S4 method for signature 'features'
```

```
getFeatureQualityNames(obj, scores = FALSE, totScore = TRUE)

## S4 replacement method for signature 'features'
analysisInfo(obj) <- value

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

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

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

## S4 method for signature 'features'
fromIMS(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,
  IMSRangeParams = NULL,
  qualityRange = NULL,
  negate = FALSE
)

## S4 method for signature 'features,ANY,missing,missing'
x[i, j, ..., ni, reorder = FALSE, 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,
    featureQualities = NULL,
    EICParams = getDefEICParams(window = 0),
    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,
  groupBy = NULL,
  showLegend = TRUE,
  xlim = NULL,
  ylim = NULL,
  ...
)

## S4 method for signature 'features'
plotBPCs(
  obj,
  retentionRange = NULL,
  MSLevel = 1,
  retMin = FALSE,
  title = NULL,
  groupBy = NULL,
  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, reorder = FALSE, 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 'featuresPiek'
delete(obj, i = NULL, j = NULL, ...)

## S4 replacement method for signature 'featuresXCMS'
analysisInfo(obj) <- value

## 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
df	If TRUE then the returned value is a <code>data.frame</code> , otherwise a <code>data.table</code> .
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.
value	A <code>data.frame</code> or <code>data.table</code> with the new analysis information.
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.
IMSRangeParams	(IMS workflow) A list with parameters to be used for filtering IMS range data. See getIMSRangeParams for details and how to make such a parameter list.

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 getFeatureQualityNames function can be used to obtain valid names). Example: <code>qualityRange=list(ModalityScore=c(0.3, Inf), SymmetryScore=c(0.5, Inf))</code> . Set to NULL to ignore.
negate	If set to TRUE then filtering operations are performed in opposite manner.
i, j	For <code>[]</code> : A numeric or character value which is used to select analyses by their index or name, respectively (for the order/names see <code>analyses()</code>). For <code>:</code> : Can also be logical to perform logical selection (similar to regular vectors). If missing all analyses are selected. For <code>[]</code> : should be a scalar value. For delete: The data to remove from. <code>i</code> are the analyses as numeric index, logical or character, <code>j</code> the features as numeric index (row) of the feature. If either is NULL then data for all is removed. <code>j</code> may also be a function: it will be called for each analysis, with the feature table (a <code>data.table</code>), the analysis name and any other arguments passed as <code>...</code> 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 <code>j</code> . For <code>plotTICs</code> and <code>plotBPCs</code> : further arguments passed to plot . For sets workflow methods: further arguments passed to the base features method.
ni	Optional argument. An expression used for subsetting the analyses. The analysis information is first subset and the remaining rows are used to determine for which analyses the results should be kept. The unevaluated <code>ni</code> expression is used to set the <code>i</code> argument of the subset operator of data.table , which therefore brings the advanced subsetting capabilities of data.table (see the data.table documentation for more details). For instance, <code>fList[replicate == "standard"]</code> would subset all analyses assigned with the replicate "standard".
reorder	If TRUE then the order of the analyses is changed to match the order of the <code>i</code> argument. (sets workflow) If the <code>sets</code> argument is specified (and <code>i</code> is not) then the order of sets is changed instead.
drop	Ignored.
name	The analysis name (partially matched).
weights	A named numeric vector that defines the weight for each score to calculate the <code>totalScore</code> . The names of the vector follow the score names. Unspecified weights are defaulted to '1'. Example: <code>weights=c(ApexBoundaryRatioScore=0.5, GaussianSimilarityScore=2)</code> .
flatnessFactor	Passed to MetaClean as the <code>flatness.factor</code> argument to calculateJaggedness and calculateModality .

featureQualities	Specifies which feature qualities to calculate. Can be NULL (default, calculates all qualities), a character vector with names of qualities to calculate (e.g., <code>c("FWHM2Base", "Symmetry")</code>), or a list of custom quality definitions. See the featureQualities function for more details.
EICParams	A named list with parameters used for extracted ion chromatogram (EIC) creation. See the EIC parameters documentation for more details.
parallel	If set to TRUE then code is executed in parallel through the future 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 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.
groupBy	Specifies how results are grouped in the plot. Should be a name of a column in the analysis information table which is used to make analysis groups (e.g. "replicate"), or "fGroups" to group by feature group. Set to NULL for no grouping.
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`: The [analysis information](#) of this features object.

`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
- `getFeatureQualityNames(features)`: Returns the present chromatographic peak quality and score names for features.

- `analysisInfo(features) <- value`: Modifies analysis information
- `analyses(features)`: returns a character vector with the names of the analyses for which data is present in this object.
- `replicates(features)`: returns a character vector with the names of the replicates for which data is present in this object.
- `hasIMS(features)`: Returns TRUE if the features object has mobility information.
- `fromIMS(features)`: Returns TRUE if the features object was directly created from IMS data.
- `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. Please see the [featureQualities](#) function and **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` A `data.table` with the [analysis information](#). Use the `analysisInfo` method for access.

`featureQualityNames` Character vector with the names of the chromatographic peak quality metrics that are present.

`hasIMS` A logical that is TRUE if the features object contain mobility/CCS information. Use the `hasIMS` method for access.

`fromIMS` A logical that is TRUE if the features object was directly created from IMS data (*i.e.* direct mobility assignment workflow). Use the `fromIMS` method for access.

S4 class hierarchy

- [workflowStep](#)
 - [features](#)
 - * [featuresSet](#)

- * [featuresUnset](#)
- * [featuresFromFeatGroups](#)
- * [featuresConsensus](#)
- * [featuresBruker](#)
- * [featuresEnviPick](#)
- * [featuresKPIC2](#)
- * [featuresOpenMS](#)
- * [featuresPiek](#)
- * [featuresSAFD](#)
- * [featuresSIRIUS](#)
- * [featuresTable](#)
- * [featuresBrukerTASQ](#)
- * [featuresXCMS](#)
- * [featuresXCMS3](#)

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by `calculatePeakQualities` and TIC/BPC related functions to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

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.
- **Important:** the `mzRange`, `mzDefectRange` and `IMSRangeParams` filters use neutral feature masses, whereas non-sets workflows use m/z values. Hence, adjust accordingly to avoid (slightly) different results!

Note

For `calculatePeakQualities`: sometimes **MetaClean** may return NA for the *Gaussian Similarity* and *Symmetry* metrics, 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	<i>Finding features</i>
--------------	-------------------------

Description

Automatically find features.

Usage

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

Arguments

analysisInfo	A data.frame (or data.table) with Analysis information .
algorithm	A character string describing the algorithm that should be used: "bruker", "openms", "xcms", "xcms3", "envipick", "sirius", "kpic2", "safd", "piek"
...	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](#), [findFeaturesPiek](#)

findFeaturesBruker	<i>Find features using Bruker DataAnalysis</i>
--------------------	--

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 (or data.table) 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 <i>always</i> , 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 DataAnalysis 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 (or data.table) with Analysis information .
...	Further parameters passed to enviPickwrap .
parallel	If set to TRUE then code is executed in parallel through the future 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	<i>Find features using KPIC2</i>
-------------------	----------------------------------

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 (or data.table) 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 future 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	<i>Find features using OpenMS</i>
--------------------	-----------------------------------

Description

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

Usage

```

findFeaturesOpenMS(
  analysisInfo,
  noiseThrInt = 1000,
  chromSNR = 3,
  chromFWHM = 5,
  mzPPM = defaultLim("mz", "medium_rel"),
  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,
  useFFMIntensities = FALSE,
  verbose = TRUE
)

```

Arguments

analysisInfo	A data.frame (or data.table) 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_out, 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 ¹³ C 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.
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 patRoanExt or configuring the path of the binaries with the patRoan.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 with useFFMIntensities=FALSE uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and [patRoön 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.

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by findFeaturesOpenMS with useFFMIntensities=FALSE to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

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 (2026). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.1, <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.

See Also

[findFeatures](#) for more details and other algorithms.

findFeaturesPiek	<i>Find features using piek</i>
------------------	---------------------------------

Description

Uses the piek algorithm to find features.

Usage

```
findFeaturesPiek(
  analysisInfo,
  genEICParams = getPiekEICParams(),
  peakParams = getDefPeakParams("chrom", "piek"),
  IMS = FALSE,
  suspects = NULL,
  adduct = NULL,
  assignMethod = "basepeak",
  assignRTWindow = defaultLim("retention", "very_narrow"),
  rtWindowDup = defaultLim("retention", "narrow"),
  mzWindowDup = defaultLim("mz", "medium"),
  mobWindowDup = defaultLim("mobility", "medium"),
  minPeakOverlapDup = 0.25,
  minIntensityIMS = 25,
  EICBatchSize = Inf,
  keepDups = FALSE,
  verbose = TRUE
)

getPiekEICParams(..., IMS = getLimIMS())
```

Arguments

analysisInfo	A data.frame (or data.table) with Analysis information .
genEICParams	A list of parameters for the EIC generation. See the EIC generation parameters section below. The getPiekEICParams function is used to generate the parameter list.
peakParams	A list of parameters for the peak detection. See getDefPeakParams for details.
IMS	A character that specifies for which IMS instrument defaults are returned. Should be "bruker" or "agilent". Defaults to what is specified in limits .
suspects	<p>The suspect list to be used for suspect pre-filtering of EIC bins. See suspect screening for details on the suspect list format and EIC generation parameters to enable suspect filtering.</p> <p>NOTE: Suspect matching can only be performed by mobilities and not CCS values. The assignMobilities method should be used to convert any CCS data in advance.</p>
adduct	An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+". Only needs to be specified if suspects is set.
assignMethod	Should be "basepeak" or "weighted.mean". This parameter sets how measured <i>m/z</i> or mobilities across the EIC datapoints are handled for feature assignment. If assignMethod="basepeak", then the value of the base peak (=highest intensity peak) from each EIC datapoint is taken. If assignMethod="weighted.mean" then the intensity weighted mean is calculated of the values that fall within the EIC bin.

assignRTWindow	<p>The retention time window (+/- seconds) used for aggregating EIC datapoints to assign feature m/z and mobility data, using an intensity weighted mean. The maximum window is always bound by the feature retention time range. Increasing this number may improve accuracy by averaging more points. However, decreasing the window may reduce inaccuracies due to inclusion of data from closely eluting features (with similar m/z and mobility) or noisy data from the chromatographic peak extremes. If assignRTWindow=0 then only the EIC data-point at the feature retention time is used.</p> <p>The assignment window is automatically adjusted for the values set for sumWindowMZ and sumWindowMob (see EIC generation parameters).</p>
rtWindowDup, mzWindowDup, mobWindowDup	<p>The retention time (seconds), m/z and mobility windows used to identify duplicate (redundant) features detected in multiple EIC bins. These values default to defaultLim("retention", "very_narrow"), defaultLim("mz", "medium") and defaultLim("mobility", "medium"), respectively (see limits).</p>
minPeakOverlapDup	<p>The minimum overlap (fraction between 0 and 1) in retention time between two features to be considered a duplicate.</p>
minIntensityIMS	<p>(IMS workflow) Raw intensity threshold for IMS data. This is primarily intended to speed up raw data processing.</p>
EICBatchSize	<p>The number of EICs to be processed in a single batch. Decreasing this number will reduce memory usage, at the cost of speed. Set to Inf to process all EICs in a single batch.</p>
keepDups	<p>Set to TRUE to keep duplicate features and features with non-centered m/z or mobility values. This is primarily intended for debugging, but can be useful to investigate why features are missing or optimize tolerance windows for duplicate feature detection.</p>
verbose	<p>If set to FALSE then no text output is shown.</p>
...	<p>Any additional parameters to be set in the returned parameter list. These will override the defaults. See the EIC generation parameters section for details.</p>

Details

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

The `piek` algorithm extends and improves on the simple and fast feature detection algorithm introduced by Dietrich C, Wick A, Ternes TA (2021). "Open-source feature detection for non-target LC-MS analytics." *Rapid Communications in Mass Spectrometry*, **36**(2). ISSN 1097-0231, doi:10.1002/rcm.9206, <http://dx.doi.org/10.1002/rcm.9206>. This algorithm first forms extracted ion chromatograms (EICs) and subsequently performs automatic peak detection to generate features. The `piek` algorithm introduces the following improvements and changes:

- Support for IMS-HRMS workflows.
- The `msdata` interface is used to efficiently form EICs from the raw data. All the file formats and types can be used that are supported by `msdata`. This includes IMS data, even if not used

for feature detection, which allows the use of IMS data directly in non-IMS or [post mobility assignment](#) workflows.

- The EIC binning approach can be extended with the mobility dimension to support [direct mobility assignment](#) workflows.
- The EIC bins can be filtered with suspect or MS2 data to speed up feature detection.
- Several filters are available to eliminate EICs with are likely devoid of any signal of interest.
- The original peak detection algorithm was further optimized or can be be exchanged with others: see [getDefPeakParams](#) for details.
- Several filters are available to improve the data and reduce redundancy:
 - The original redundancy detection, which performs a second feature detection with EIC bins that are shifted by 50% width and eliminates features with m/z values outside the center of any bin, was extended for IMS support.
 - Redundant features across bins are eliminated if with close retention time, m/z, mobility and chromatographic overlap. The most intense feature is kept.
 - Data from suspects or MS2 precursors that was used to pre-filter EICs, can also be used to filter the final feature list.
- Various small bug fixes and improvements for the original code.
- The output feature tables contain raw intensities/areas and those subtracted by the estimated noise level (`intensity`, `intensitySub`, `area` and `areaSub` columns, respectively) and the estimated signal to noise (`signalToNoise` column).

Value

`getPiekEICParams` returns a list of parameters for the EIC generation, which is used to set the `genEICParams` argument to `findFeaturesPiek`.

IMS workflows

If IMS data is used to resolve features (`IMS=TRUE`), a 'pre-check' is performed to avoid excessive numbers of two-dimensional bins for EIC formation and peak detection. These EICs are formed by only considering the m/z dimension, and subsequently filtered by the parameters described in the EIC generation

parameters section. The final EICs for feature detection are then only formed if they have m/z data that was not removed during the pre-check.

The m/z and mobility data from IMS-HRMS data is typically not or partially centroided. The feature m/z and mobility values are derived from m/z or mobility *versus* intensity profiles. The profiles are generated for each EIC timepoint, and the value at the maximum intensity or intensity weighted mean of the profile is used to derive the intermediate values (configured by `assignMethod`). Several parameters exist to improve the profile data (see next section).

EIC generation parameters

The `genEICParams` argument to `findFeaturesPiek` configures the generation of EICs. The `getPiekEICParams` function should be used to generate the parameter list.

The following general parameters exist:

- **filter** Controls the pre-filtering of EIC bins with m/z data. Should be "none" (no filtering), "suspects" (filter with suspect data) or "ms2" (filter with data from precursors detected in a data-dependent MS/MS experiment).
- **mzRange,mzStep** Configures the formation of m/z bins. **mzRange** is a numeric vector of length two that specifies the min/max m/z range. **mzStep** specifies the bin widths.
- **retRange** A numeric vector of length two that specifies the retention time range for the EICs. Data outside this range is excluded. Set to NULL to use the full range.
- **gapFactor** A numeric that configures gap filling for EICs. See [getDefEICParams](#) for further details.
- **minEICIntensity** The minimum intensity of the highest data point in the EIC. Used to filter EICs.
- **minEICAdjTime,minEICAdjPoints,minEICAdjIntensity** The EIC should have at least a continuous signal of **minEICAdjTime** seconds and **minEICAdjPoints** data points, where the continuity is defined by data points with an intensity of at least **minEICAdjIntensity** high. Set **minEICAdjTime** or **minEICAdjPoints** to zero to disable continuity checks for time or data points, respectively. Set **minEICAdjIntensity** to zero to completely disable continuity checks.
- **topMostEICMZ** Only keep this number of top-most intense EICs. The intensity is derived from the data point with the highest intensity in the EIC. Set to zero to always select all EICs. For IMS workflows, this parameter is *only* used to limit the number of EICs resulting from the 'pre-check' in the m/z dimension.

The following parameters are specifically used for IMS workflows:

- **filterIMS** Similar to the **filter** parameter, but controls how mobility data is used for pre-filtering of EIC bins.

Different values for **filter** and **filterIMS** can be specified:

- **filter**="none" and **filterIMS**="none"
- **filter**="suspects" and **filterIMS**="suspects"
- **filter**="suspects" and **filterIMS**="none" (only use m/z filtering)
- **filter**="ms2" and **filterIMS**="ms2"
- **filter**="ms2" and **filterIMS**="none"

Currently only Bruker DDA-PASEF experiments provide the data needed for "ms2" filtering.

- **mobRange,mobStep** Equivalent to **mzRange** and **mzStep**, but for ion mobility binning.
- **sumWindowMZ,sumWindowMob** The retention time window (+/- s) used to sum adjacent data-points for the determination of intermediate EIC m/z and mobility values. This data is aggregated to determine the final feature values (see also the **assignRTWindow** argument). Set to '0' to not sum any adjacent timepoints. Larger values can generally improve accuracy for noisy data (e.g. from TIMS), but care must be taken to stay below the expected minimum chromatographic peak width to avoid inclusion of data from other features. Defaults to `defaultLim("retention", "very_narrow")` (see [limits](#)).
- **smoothWindowMZ,smoothWindowMob** The window size used to perform centered moving average smoothing on intensity data of the m/z and mobility profiles used to determine intermediate EIC values. Smoothing of noisy data (e.g. TIMS) is highly recommended to improve accuracy and consistency. Set to 0 to disable smoothing.

- `smoothExtMZ,smoothExtMob` The m/z or mobility window to extend the smoothing at the edges of the EIC bin. This is recommended to improve smoothing, *e.g.* when the peak profile is only partially captured in the bin. Defaults to the bin width, *i.e.* data from an adjacent bin on each side is additionally included for smoothing. The final smoothed data is only taken from the actual EIC bin. Set to 0 to disable extension.
- `saveMZProfiles,saveEIMs` Set to TRUE to save the m/z and mobility profiles for each feature. Only the profiles at the feature retention time is saved. This can be useful for debugging or parameter optimization, but will increase memory usage and processing times.
- `topMostEICMob` Equivalent to `topMostEICMZ`, used to reduce the final two-dimensional EIC bins with m/z and mobility information.
- `minEICsIMSPreCheck` Only perform the m/z pre-check if the number of two-dimensional EIC bins is at least `minEICsIMSPreCheck`.

The following parameters are specifically for when suspect data is used to pre-filter EIC bins:

- `rtWindow,mzwindow,mobWindow`: The retention time, m/z and mobility tolerance windows for suspect data. These are used for:
 1. Pre-filtering of EIC bins with suspect data, *i.e.* larger tolerances will lead to more EIC bins being kept. (only applicable for `mzWindow` and `mobWindow`).
 2. Matching the final features to suspect data. `rtWindow=Inf` can be used to disable retention time matching.

Defaults to `defaultLim("retention", "medium")`, `defaultLim("mz", "medium")` and `defaultLim("mobility", "medium")`, see [limits](#).

- `skipInvalid,prefCalcChemProps,neutralChemProps` Controls preparing the suspect list data. See [screenSuspects](#).

The following parameters are specifically for when MS2 data is used to pre-filter EICs:

- `rtWindow` Eliminates any features without an MS/MS spectrum within this retention time window. Set `rtWindow=Inf` to disable this filter. Defaults to `defaultLim("retention", "very_narrow")` (see [limits](#)).
- `mzIsoWindow` The maximum m/z window considered for MS/MS precursors that were isolated by DDA. These m/z isolation windows are used to pre-filter EICs and match the final features. Setting `mzIsoWindow` to a value lower than typical instrument isolation windows will make feature detection more specific, as features need to be more close to the triggered DDA precursor m/z values. In contrast, larger values for `mzIsoWindow` allows to include features that were not specifically targeted by DDA, but may still have MS/MS data as their m/z could still fall within the MS/MS isolation window. The effective window used will never exceed the instrumental isolation window. Setting `mzIsoWindow=Inf` will always use instrumental windows.
- `mobWindow` The mobility tolerance window to match DDA MS/MS precursors in IMS workflows. Used for pre-filtering EICs and the final features. To match DDA precursor data, the measured mobility range of the corresponding MS/MS data is used as the mobility window. This window is then adjusted to be at least \pm `mobWindow`. Defaults to `defaultLim("mobility", "medium")` (see [limits](#)).
- `minTIC` The minimum total ion current (TIC) signal for an MS/MS spectrum to be considered. Can be increased to eliminate features with low intensity MS/MS data.

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by findFeaturesPiek to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

The use of profile m/z HRMS data (not IMS-HRMS) is currently not supported.

References

There are no references for Rd macro \insertAllCites on this help page.

See Also

[findFeatures](#) for more details and other algorithms.

findFeaturesSAFD	<i>Find features using SAFD</i>
------------------	---------------------------------

Description

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

Usage

```
findFeaturesSAFD(  
  analysisInfo,  
  prefCentroid = FALSE,  
  mzRange = c(0, 400),  
  maxNumbIter = 1000,  
  maxTPeakW = 300,  
  resolution = 30000,  
  minMSW = 0.02,  
  RThreshold = 0.75,  
  minInt = 2000,  
  sigIncThreshold = 5,  
  S2N = 2,  
  minPeakWS = 3,  
  centroidMethod = "RFM",  
  centroidDM = 0.005,  
  verbose = TRUE  
)
```

Arguments

analysisInfo	A data.frame (or data.table) with Analysis information .
prefCentroid	Set to TRUE to prefer centroided data over other MS data specified in analysisInfo. NOTE: if prefCentroid=FALSE but the package option 'patRoön.MS.preferIMS=TRUE' (see msdata), then centroided data will still be preferred over IMS data.

mzRange	The m/z window to be imported.
maxNumIter, maxTPeakW, resolution, minMSW, RThreshold, minInt, sigIncThreshold, S2N, minPeakWS	Parameters directly passed to the safd_s3D function.
centroidMethod, centroidDM	Passed to the safd_s3d_cent function (method and mdm arguments, respectively).
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 with SAFD.

As of **patRoön** '3.0', findFeaturesSAFD uses the [msdata](#) interface instead of the **MS_Import.jl** Julia package to read HRMS data. This means that **MS_Import.jl** does not need to be installed, and all file formats supported by [msdata](#) are also supported for SAFD feature detection. This includes IMS-HRMS data, however, in that case IMS resolved spectra are summed and the IMS dimension is removed to make the data compatible for SAFD.

The SAFD algorithm was primarily developed to detect features in profile m/z data, but centroided data is also supported. To use profile data, ensure that the paths are correctly set up in the [analysisInfo](#). Furthermore, when using profile data you probably also need to specify centroided data in the analysisInfo, as *e.g.* [generateMSPeakLists](#) currently does not support profile data. If IMS-HRMS data is used it is treated as profile data, as this data is typically not or partially centroided (generateMSPeakLists supports IMS-HRMS data directly).

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 [patRoön 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](https://doi.org/10.1021/acs.analchem.9b02422).

See Also

[findFeatures](#) for more details and other algorithms.

findFeaturesSIRIUS	<i>Find features using SIRIUS</i>
--------------------	-----------------------------------

Description

Uses **SIRIUS** to find features.

Usage

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

Arguments

analysisInfo	A data.frame (or data.table) 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 algorithm 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 [patRoön 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	<i>Find features using XCMS (old interface)</i>
------------------	---

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 (or data.table) 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	<i>Find features using XCMS (new interface)</i>
-------------------	---

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 (or data.table) 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	<i>Formula annotations class</i>
----------------	----------------------------------

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,
    normalized = "multiple",
    specSimParams = getDefSpecSimParams(),
    mincex = 0.9,
    xlim = NULL,
    ylim = NULL,
    showLegend = TRUE,
    ...
)

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

## S4 method for signature 'formulas'
consensus(
    obj,
```

```

    ...,
    MSPeakLists,
    specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
    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,
  normalized = "multiple",
  specSimParams = getDefSpecSimParams(),
  mincex = 0.9,
  xlim = NULL,
  ylim = NULL,
  showLegend = TRUE,
  perSet = TRUE,
  mirror = TRUE,
  ...
)

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

## S4 method for signature 'formulasSet'
consensus(
  obj,

```



```

    ...,
    MSPeakLists,
    specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
    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

<code>obj, x, object</code>	The formulas object.
<code>features</code>	If TRUE returns formula data for features, otherwise for feature groups.
<code>i, j</code>	For <code>[]</code> : If both <code>i</code> and <code>j</code> are specified then <code>i</code> specifies the analysis and <code>j</code> the feature group of the feature for which annotations should be returned. Otherwise <code>i</code> specifies the feature group for which group annotations should be returned. <code>i/j</code> can be specified as integer index or as a character name. Otherwise passed to the featureAnnotations method.
<code>...</code>	For <code>plotSpectrum</code> : Further arguments passed to plot . For <code>delete</code> : passed to the function specified as <code>j</code> . For <code>consensus</code> : Any further (and unique) formulas objects. For sets workflow methods: further arguments passed to the base formulas method.
<code>fGroups, fragments, countElements, countFragElements, OM</code>	Passed to the featureAnnotations method.
<code>normalizeScores</code>	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`).

<code>excludeNormScores</code>	<p>A character vector specifying any compound scoring names that should <i>not</i> be normalized. Set to <code>NULL</code> to normalize all scorings. Note that whether any normalization occurs is set by the <code>excludeNormScores</code> argument.</p> <p>For compounds: By default <code>score</code> and <code>individualMoNAScore</code> are set to mimic the behavior of the MetFrag web interface.</p>
<code>average</code>	<p>If set to <code>TRUE</code> 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 <code>TRUE</code> disables reporting of most other data.</p>
<code>index</code>	<p>The candidate index (row). For <code>plotSpectrum</code> two indices can be specified to compare spectra. In this case <code>groupName</code> and <code>analysis</code> (if not <code>NULL</code>) should specify values for the spectra to compare.</p>
<code>groupName</code>	<p>The name of the feature group for which a plot should be made. To compare spectra, two group names can be specified.</p>
<code>analysis</code>	<p>The name of the analysis for which a plot should be made. If <code>NULL</code> then data from the feature group averaged peak list is used. When comparing spectra, either <code>NULL</code> or the analyses for both spectra should be specified.</p>
<code>MSPeakLists</code>	<p>The MSPeakLists object with relevant spectral data.</p>
<code>onlyAnnotated</code>	<p>Set to <code>TRUE</code> to filter out any peaks that could not be annotated.</p>
<code>title</code>	<p>The title of the plot. If <code>NULL</code> a title will be automatically made.</p>
<code>normalized</code>	<p>Controls intensity normalization. Should be <code>FALSE</code> (don't normalize), <code>TRUE</code> (normalize) or "multiple" (only normalizes if multiple spectra are plotted).</p>
<code>specSimParams</code>	<p>A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.</p>
<code>mincex</code>	<p>The formula annotation labels are automatically scaled. The <code>mincex</code> argument forces a minimum <code>cex</code> value for readability.</p>
<code>xlim, ylim</code>	<p>Sets the plot size limits used by plot. Set to <code>NULL</code> for automatic plot sizing.</p>
<code>showLegend</code>	<p>Set to <code>TRUE</code> to show a legend.</p>
<code>absMinAbundance, relMinAbundance</code>	<p>Minimum absolute or relative ('0-1') abundance across objects for a result to be kept. For instance, <code>relMinAbundance=0.5</code> 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 <code>uniqueFrom</code> is not <code>NULL</code>.</p>
<code>uniqueFrom</code>	<p>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 <code>uniqueFrom</code> to a logical (values are recycled), numeric (select by index) or a character (as obtained with <code>algorithm(obj)</code>). For logical and numeric values the order corresponds to the order of the objects given for the consensus. Set to <code>NULL</code> to ignore.</p>

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 annotation 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 consensus 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.

`MS2QuantMeta` (**sets workflow**) A named list with for each set the metadata from `MS2Quant` filled in by `predictRespFactors`.

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 `updateConsensus`, 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	<i>Grouping feature groups in components</i>
--------------------	--

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.

IMS workflows

The componentization algorithm is not aware of the IMS dimension. For this reason, no IMS feature groups will be considered for componentization, and direct IMS workflows (see [assignMobilities](#)) are currently not supported.

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 = defaultLim("mz", "medium"),  
  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>i.e.</i> 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 assignments 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 <i>preferential adducts</i> . 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 future 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.

IMS workflows

The componentization algorithm is not aware of the IMS dimension. For this reason, no IMS feature groups will be considered for componentization, and direct IMS workflows (see `assignMobilities`) are currently not supported.

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](https://doi.org/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,
  IMS = "maybe"
)
```

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](#).

IMS (IMS workflow) Specifies which feature groups are considered for componentization in IMS workflows. The following options are valid:

- "both": Selects IMS and non-IMS features.
- "maybe": Selects non-IMS features and IMS features without assigned IMS precursor.
- FALSE: Selects only non-IMS features.
- TRUE: Selects only IMS features.

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](#).

IMS workflows

In IMS workflows with post mobility assignment (see [assignMobilities](#)) it may be necessary to call [expandForIMS](#) when componentization was performed *prior* to mobility assignments, see its documentation for more details.

If mobilities were already assigned prior to componentization, then the IMS argument selects which feature groups are subjected to componentization. Data for IMS feature groups that were not considered (*i.e.* when IMS is FALSE or "maybe"), will be expanded similarly as is done by [expandForIMS](#).

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 = defaultLim("retention", "wide"),
  absMzDev = defaultLim("mz", "narrow"),
  absMzDevLink = defaultLim("mz", "medium"),
  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 (analyses within each replicate are averaged prior to detection). Then, homologous series across replicates are merged in case of full overlap or when merging of partial overlapping series causes no conflicts.

Value

The generated components 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.

IMS workflows

The componentization algorithm is not aware of the IMS dimension. For this reason, no IMS feature groups will be considered for componentization, and direct IMS workflows (see [assignMobilities](#)) are currently not supported.

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 = defaultLim("retention", "very_narrow"),  
  absMzDev = defaultLim("mz", "medium"),  
  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 <code>algorithm:MetaboliteFeatureDeconvolution</code> options.
chargeSpan	The maximum charge span for a single analyte. Corresponds to <code>algorithm:MetaboliteFeatureDeconvolution</code> .
qTry	Sets how charges are determined. Corresponds to <code>algorithm:MetaboliteFeatureDeconvolution:q_try</code> . 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, <i>e.g.</i> <code>potentialAdducts=c("[M+H]+" = 0.8, "[M+Na]+" = 0.2)</code> . 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 <i>not</i> with potentialAdducts), <i>e.g.</i> if chargeMax=2 then both [M+H]+ and [2M+H]2+ may be considered. Please see the <code>algorithm:MetaboliteFeatureDeconvolution</code> option of MetaboliteAdductDecharger for more details. If NULL then the a default is chosen with defaultOpenMSAdducts (which is <i>not</i> 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 <code>sets(fGroups)</code> . Example: <code>potentialAdducts=list(positive=c("[M+H]+" = 0.8, "[M+Na]+" = 0.2), negative=c("[M-H]-" = 0.8, "[M-H2O-H]-" = 0.2))</code>
minRTOverlap, retWindow	Sets feature retention tolerances when grouping features. Sets the <code>algorithm:MetaboliteFeatureDeconvolution</code> and " <code>algorithm:MetaboliteFeatureDeconvolution:retention_max_diff</code> " options.
absMzDev	Maximum absolute <i>m/z</i> deviation. Sets the <code>algorithm:MetaboliteFeatureDeconvolution:mass_max_dev</code> 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 assignments to the features within a feature group are equal and at least one of those adducts is a preferential adduct (<code>prefAdducts</code> argument), then only the features with (the lowest ranked) preferential adduct are considered. In all other cases or when <code>adductConflictsUsePref=FALSE</code>

	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 <i>preferential adducts</i> . 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 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.

IMS workflows

The componentization algorithm is not aware of the IMS dimension. For this reason, no IMS feature groups will be considered for componentization, and direct IMS workflows (see [assignMobilities](#)) are currently not supported.

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 [patRoos 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 = defaultLim("mz", "narrow"),
relMzDev = defaultLim("mz", "narrow_rel"),
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 <code>mzabs.error</code> argument to do.findmain .
relMzDev	Maximum relative mass deviation (PPM). Sets the <code>ppm.error</code> argument to do.findmain .
minSize	The minimum size of a component. Smaller components than this size will be removed. See note below. Sets the <code>minModuleSize</code> 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.

IMS workflows

The componentization algorithm is not aware of the IMS dimension. For this reason, no IMS feature groups will be considered for componentization, and direct IMS workflows (see [assignMobilities](#)) are currently not supported.

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,
  IMS = "maybe"
)
```

Arguments

- | | |
|--|---|
| <code>fGroups</code> | <code>featureGroups</code> object for which components should be generated. |
| <code>...</code> | Any parameters to be passed to the selected component generation algorithm. |
| <code>MSPeakLists</code> | The <code>MSPeakLists</code> object for the given feature groups that should be used for MS spectral similarity calculations. |
| <code>method</code> | Clustering method that should be applied (passed to <code>fastcluster::hclust</code>). |
| <code>specSimParams</code> | A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details. |
| <code>maxTreeHeight</code> , <code>deepSplit</code> , <code>minModuleSize</code> | Arguments used by <code>cutreeDynamicTree</code> . |
| <code>IMS</code> | (IMS workflow) Specifies which feature groups are considered for componentization in IMS workflows. The following options are valid: <ul style="list-style-type: none">• "both": Selects IMS and non-IMS features.• "maybe": Selects non-IMS features and IMS features without assigned IMS precursor.• FALSE: Selects only non-IMS features.• TRUE: Selects only IMS features. |

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`.

IMS workflows

In IMS workflows with post mobility assignment (see `assignMobilities`) it may be necessary to call `expandForIMS` when componentization was performed *prior* to mobility assignments, see its documentation for more details.

If mobilities were already assigned prior to componentization, then the IMS argument selects which feature groups are subjected to componentization. Data for IMS feature groups that were not considered (*i.e.* when IMS is FALSE or "maybe"), will be expanded similarly as is done by `expandForIMS`.

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,
  TPs = NULL,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  ignoreParents = FALSE,
  minRTDiff = 20,
  specSimParams = getDefSpecSimParams(),
  IMS = "maybe"
)

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

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 <i>must</i> be a subset of the same featureGroups object.
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.
ignoreParents	If TRUE then feature groups present in both fGroups and fGroupsTPs are not considered as TPs.
minRTDiff	Minimum retention time (in seconds) difference between the parent and a TP to calculate the retention order direction .
specSimParams	A named list with parameters that influence the calculation of MS spectra similarities. See the spectral similarity parameters documentation for more details.
IMS	(IMS workflow) Specifies which feature groups are considered for componentization in IMS workflows. The following options are valid: <ul style="list-style-type: none"> • "both": Selects IMS and non-IMS features. • "maybe": Selects non-IMS features and IMS features without assigned IMS precursor. • FALSE: Selects only non-IMS features. • TRUE: Selects only IMS features.

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 parents and/or TPs may need to be performed in advance to facilitate linkage. This depends on the algorithm that was used to generate the TPs:

- the parents need to be screened for all algorithms except logic ([generateTPsLogic](#))
- the TPs need to be screened for all algorithms except ann_form and ann_comp ([generateTPsAnnForm](#) and [generateTPsAnnComp](#)).

See [screenSuspects](#) to perform the screening and [convertToSuspects](#) to create the suspect list. To include parents make sure to set includeParents=TRUE when calling convertToSuspects or first screen for the parents and then amend the screening object with TP screening results by setting amend=TRUE to screenSuspects. 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.

Result columns

During componentization, several characteristics are calculated which may be useful for post-processing. These can be obtained with e.g. [as.data.table](#) or [componentTable](#). The properties are either reported for each feature group in a component, or for each candidate of a feature group in a component (only if TPs was set).

The following properties may be reported for each feature group:

- 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).
- totalFragmentMatches The total number of MS/MS fragment annotations that overlap between *all* feature annotation candidates for the TP feature group and the feature annotations specifically for the parent (based on the assigned fragment formula). If both the formulas and compounds arguments are specified then the annotation data is pooled prior to calculation. Each unique match is only counted once.
- totalNeutralLossMatches As totalFragmentMatches, but counting overlapping neutral loss formulae.
- retDir,TP_retDir The [retention order direction](#) derived from the feature groups (retDir) or the (expected) value from TP data (TP_retDir).
- retDiff,mzDiff, The retention time and *m/z* difference between the parent and TP.

The candidate specific properties are stored inside the candidates column in component tables, and can be obtained with [as.data.table](#) by setting candidates=TRUE. The following properties may be present:

- `fragmentMatches,neutralLossMatches` As `totalFragmentMatches` and `totalNeutralLossMatches`, but only considering the feature annotations specifically for this candidate.
- `formulaDiff` The formula difference between the parent and TP (if formula data is available).
- `TPScore,annSim,fitFormula,fitCompound,simSusps`: TP scoring properties, see [generateTPsAnnForm](#) and [generateTPsAnnComp](#).

IMS workflows

In IMS workflows with post mobility assignment (see [assignMobilities](#)) it may be necessary to call [expandForIMS](#) when componentization was performed *prior* to mobility assignments, see its documentation for more details.

If mobilities were already assigned prior to componentization, then the IMS argument selects which feature groups are subjected to componentization. Data for IMS feature groups that were not considered (*i.e.* when IMS is FALSE or "maybe"), will be expanded similarly as is done by [expandForIMS](#).

NOTE: IMS expansion by [expandForIMS](#) only expands results for TP candidates, *i.e.* no new components from parents assigned to IMS feature groups will be added.

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.

<code>generateCompounds</code>	<i>Automatic compound annotation</i>
--------------------------------	--------------------------------------

Description

Automatically perform chemical compound annotation for feature groups.

Usage

```
generateCompounds(  
  fGroups,  
  MSPeakLists,  
  algorithm,  
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),  
  ...  
)
```

```
)

## S4 method for signature 'featureGroups'
generateCompounds(
  fGroups,
  MSPeakLists,
  algorithm,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  ...
)
```

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"
specSimParams	A named list with parameters that influence the calculation of the annotation similarity . See the spectral similarity parameters documentation for more details.
...	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,  
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),  
  MSLibrary,  
  minSim = 0.75,  
  minAnnSim = minSim,  
  absMzDev = defaultLim("mz", "narrow"),  
  adduct = NULL,
```

```

    checkIons = "adduct",
    spectrumType = "MS2",
    specSimParamsLib = specSimParams,
    minIMSSpecSim = 0
)

## S4 method for signature 'featureGroupsSet'
generateCompoundsLibrary(
  fGroups,
  MSPeakLists,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  MSLibrary,
  minSim = 0.75,
  minAnnSim = minSim,
  absMzDev = defaultLim("mz", "narrow"),
  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.
specSimParams	A named list with parameters that influence the calculation of the annotation similarity . See the spectral similarity parameters documentation for more details.
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, <i>e.g.</i> "MS2"). Set to NULL to allow all spectrum types.
specSimParamsLib	Like specSimParams, but these parameters are used for the pre-treatment of library spectra (only the removePrecursor, relMinIntensity and minPeaks parameters are used).
minIMSSpecSim	(IMS workflow) If the spectrum similarity of an IMS feature group compared to its IMS precursor (see assignMobilities) is at least this value, then the IMS feature group will not be subjected to the annotation algorithm and all feature annotation properties will be copied from its precursor. This assumes that feature annotation is primarily influenced by the MS/MS spectrum, and can be used to speed up the feature annotation process. All scorings, annotation similarities etc. are copied from the IMS precursor. The fragment annotations are also copied (fragInfo result column), however, these are adjusted based on the peak list data of the IMS feature group.
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 (<i>e.g.</i> 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.

Note

The score, libMatch and annSim output columns are all equal and resemble the spectral similarity between the experimental and library spectra.

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,  
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),  
  method = "CL",  
  timeout = 300,  
  timeoutRetries = 2,  
  errorRetries = 2,  
  topMost = 100,  
  dbRelMzDev = defaultLim("mz", "narrow_rel"),  
  fragRelMzDev = defaultLim("mz", "narrow_rel"),  
  fragAbsMzDev = defaultLim("mz", "narrow"),  
  adduct = NULL,  
  database = "pubchemlite",  
  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,
    minIMSSpecSim = 0
)

## S4 method for signature 'featureGroupsSet'
generateCompoundsMetFrag(
  fGroups,
  MSPeakLists,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  method = "CL",
  timeout = 300,
  timeoutRetries = 2,
  errorRetries = 2,
  topMost = 100,
  dbRelMzDev = defaultLim("mz", "narrow_rel"),
  fragRelMzDev = defaultLim("mz", "narrow_rel"),
  fragAbsMzDev = defaultLim("mz", "narrow"),
  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.
specSimParams	A named list with parameters that influence the calculation of the annotation similarity . See the spectral similarity parameters documentation for more details.
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 'DatabaseSearchRelativeMassDeviation' 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 <i>extended</i> database that includes information for compound scoring (<i>i.e.</i> 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>i.e.</i> 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 (<i>e.g.</i> "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 significantly 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 <code>compounds</code> object obtained with <code>generateCompoundsSIRIUS</code> using the <code>identifiers</code> method. This way, only those candidates will be searched by MetFrag that were generated by SIRIUS+CSI:FingerID. Sets the 'PrecursorCompoundIDs' option.
<code>extraOpts</code>	A named list containing further settings MetFrag. See the MetFragR and MetFrag CL homepages for all available options. Set to NULL to ignore.
<code>minIMSSpecSim</code>	(IMS workflow) If the spectrum similarity of an IMS feature group compared to its IMS precursor (see assignMobilities) is at least this value, then the IMS feature group will not be subjected to the annotation algorithm and all feature annotation properties will be copied from its precursor. This assumes that feature annotation is primarily influenced by the MS/MS spectrum, and can be used to speed up the feature annotation process. All scorings, annotation similarities etc. are copied from the IMS precursor. The fragment annotations are also copied (<code>fragInfo</code> result column), however, these are adjusted based on the peak list data of the IMS feature group.
<code>setThreshold</code>	(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.
<code>setThresholdAnn</code>	(sets workflow) As <code>setThreshold</code> , but only taking into account the set data that contain annotations for the feature group of the candidate.
<code>setAvgSpecificScores</code>	(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	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](#).

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 **patRoanExt** is *not* installed then the file location must be specified as above or by setting the `patRoan.path.MetFragPubChemLite/patRoan.path.MetFragComptox` option. See the installation section in the handbook for more details.

If `database="pubchemlite"` and the local file has CCS predictions (see <https://zenodo.org/records/15311000>), then CCS values will be copied from the corresponding adduct of the feature groups (or as specified by the `adduct` argument). These can be converted to mobilities with `assignMobilities` and used for candidate filtering with `filter`.

Parallelization

`generateCompoundsMetFrag` uses multiprocessing to parallelize computations. Please see the parallelization section in the handbook for more details and [patRoan options](#) for configuration options.

When local database files are used with `generateCompoundsMetFrag` (e.g. when `database` is set to `"pubchemlite"`, `"csv"` etc.) and `'patRoan.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 (`'patRoan.path.MetFragPubChemLite'` and `'patRoan.path.MetFragCompTox'`) or made available by installing the **patRoanExt** package. Note that these files must *also* be present on the local host computer, even if it is not participating in computations.

If the compound database is *not* local, e.g. `database="pubchem"`, then parallelization is disabled to avoid connection errors that typically occur otherwise.

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.
- Schymanski EL, Kondić T, Neumann S, Thiessen PA, Zhang J, Bolton EE (2021). “Empowering large chemical knowledge bases for exposomics: PubChemLite meets MetFrag.” *Journal of Cheminformatics*, **13**(1). ISSN 1758-2946, doi:10.1186/s13321021004890, <http://dx.doi.org/10.1186/s13321-021-00489-0>.
- Elapavalore A, Ross DH, Grouès V, Aurich D, Krinsky AM, Kim S, Thiessen PA, Zhang J, Dodds JN, Baker ES, Bolton EE, Xu L, Schymanski EL (2025). “PubChemLite Plus Collision Cross Section (CCS) Values for Enhanced Interpretation of Nontarget Environmental Data.” *Environmental Science & Technology Letters*, **12**(2), 166–174. ISSN 2328-8930, doi:10.1021/acs.estlett.4c01003, <http://dx.doi.org/10.1021/acs.estlett.4c01003>.
- Ross DH, Cho JH, Xu L (2020). “Breaking Down Structural Diversity for Comprehensive Prediction of Ion-Neutral Collision Cross Sections.” *Analytical Chemistry*, **92**(6), 4548–4557. ISSN 1520-6882, doi:10.1021/acs.analchem.9b05772, <http://dx.doi.org/10.1021/acs.analchem.9b05772>.

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,  
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),  
  relMzDev = defaultLim("mz", "narrow_rel"),  
  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,
minIMSSpecSim = 0,
verbose = TRUE,
splitBatches = FALSE,
dryRun = FALSE
)

## S4 method for signature 'featureGroupsSet'
generateCompoundsSIRIUS(
  fGroups,
  MSPeakLists,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  relMzDev = defaultLim("mz", "narrow_rel"),
  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.
specSimParams	A named list with parameters that influence the calculation of the annotation similarity . See the spectral similarity parameters documentation for more details.
relMzDev	Maximum relative deviation between the measured and candidate formula <i>m/z</i> 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	<p>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.</p> <p>(sets workflow) projectPath should be a character specifying the paths for each set.</p>
elements	<p>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.</p>
profile	<p>Name of the configuration profile, for example: "qtof", "orbitrap", "fticr". Sets the '--profile' commandline option.</p>
formulaDatabase	<p>If not NULL, use a database for retrieval of formula candidates. Possible values are: "pubchem", "bio", "kegg", "hmdb". Sets the '--database' commandline option.</p>
fingerIDDatabase	<p>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.</p>
noise	<p>Median intensity of the noise (NULL ignores this parameter). Sets the '--noise' commandline option.</p>
cores	<p>The number of cores SIRIUS will use. If NULL then the default of all cores will be used.</p>
topMost	<p>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.</p>
topMostFormulas	<p>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.</p>
login, alwaysLogin	<p>Specifies if and how account logging of SIRIUS should be handled:</p> <p>login=FALSE: no automatic login is performed and the active login status is not checked.</p> <p>login="check": aborts if no active login is present.</p> <p>login="interactive": interactively ask for login (using getPass).</p> <p>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.</p> <p>if alwaysLogin=TRUE then a login is always performed, otherwise only if SIRIUS reports no active login.</p> <p>See the SIRIUS website and patRoön handbook for more information.</p>

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.
minIMSSpecSim	(IMS workflow) If the spectrum similarity of an IMS feature group compared to its IMS precursor (see assignMobilities) is at least this value, then the IMS feature group will not be subjected to the annotation algorithm and all feature annotation properties will be copied from its precursor. This assumes that feature annotation is primarily influenced by the MS/MS spectrum, and can be used to speed up the feature annotation process. All scorings, annotation similarities etc. are copied from the IMS precursor. The fragment annotations are also copied (fragInfo result column), however, these are adjusted based on the peak list data of the IMS feature group.
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 [patRoön 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 `'patRoos.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	<i>Automatic chemical formula generation</i>
------------------	--

Description

Automatically calculate chemical formulae for all feature groups.

Usage

```
generateFormulas(  
  fGroups,  
  MSPeakLists,  
  algorithm,  
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),  
  ...  
)  
  
## S4 method for signature 'featureGroups'  
generateFormulas(  
  fGroups,  
  MSPeakLists,
```

```

    algorithm,
    specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
    ...
)

```

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"
specSimParams	A named list with parameters that influence the calculation of the annotation similarity . See the spectral similarity parameters documentation for more details.
...	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).

generateFormulasGenForm

Generate formula with GenForm

Description

Uses [GenForm](#) to generate chemical formula candidates.

Usage

```

generateFormulasGenForm(fGroups, ...)

## S4 method for signature 'featureGroups'
generateFormulasGenForm(
  fGroups,
  MSPeakLists,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  relMzDev = defaultLim("mz", "narrow_rel"),
  adduct = NULL,
  elements = "CHNOP",
  hetero = TRUE,
  oc = FALSE,
  thrMS = NULL,
  thrMSMS = NULL,
  thrComb = NULL,
  maxCandidates = Inf,
  extraOpts = NULL,
  calculateFeatures = FALSE,
  featThreshold = 0,
  featThresholdAnn = 0.75,
  absAlignMzDev = defaultLim("mz", "narrow"),
  MSMode = "both",
  isolatePrec = TRUE,
  minIMSSpecSim = 0,
  timeout = 120,
  topMost = 50,
  batchSize = 8
)

## S4 method for signature 'featureGroupsSet'
generateFormulasGenForm(
  fGroups,
  MSPeakLists,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  relMzDev = defaultLim("mz", "narrow_rel"),
  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.
specSimParams	A named list with parameters that influence the calculation of the annotation similarity . See the spectral similarity parameters documentation for more details.
relMzDev	Maximum relative deviation between the measured and candidate formula <i>m/z</i> 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>i.e.</i> 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 <i>after</i> 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 <code>absAlignMzDev</code> 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 <code>isolatePrec</code> should either be a list with parameters (see the filter method for <code>MSPeakLists</code> for more details), TRUE for default parameters or FALSE for no isolation (e.g. when you already performed isolation with the <code>filter</code> method). The <code>z</code> parameter (charge) is automatically deduced from the adduct used for annotation (unless <code>isolatePrec=FALSE</code>), hence any custom <code>z</code> setting is ignored.
minIMSSpecSim	(IMS workflow) If the spectrum similarity of an IMS feature group compared to its IMS precursor (see assignMobilities) is at least this value, then the IMS feature group will not be subjected to the annotation algorithm and all feature annotation properties will be copied from its precursor. This assumes that feature annotation is primarily influenced by the MS/MS spectrum, and can be used to speed up the feature annotation process. All scorings, annotation similarities etc. are copied from the IMS precursor. The fragment annotations are also copied (<code>fragInfo</code> result column), however, these are adjusted based on the peak list data of the IMS feature group. This argument does not affect the annotation results for MS-only formulae.
timeout	Maximum time (in seconds) that a <code>GenForm</code> 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 <code>GenForm</code> commands that should be run sequentially in each parallel process. Combining commands with short runtimes (such as <code>GenForm</code>) can significantly increase parallel performance. For more information see executeMultiProcess . Note that this is ignored if <code>'patRoon.MP.method="future"'</code> .
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 <code>setThreshold</code> , 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|combm]] [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
omsm     : 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 [patRoön options](#) for configuration options.

When futures are used for parallel processing (patRoön.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,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  relMzDev = defaultLim("mz", "narrow_rel"),
  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 = FALSE,
  featThreshold = 0,
  featThresholdAnn = 0.75,
  absAlignMzDev = defaultLim("mz", "narrow"),
  minIMSSpecSim = 0,
  verbose = TRUE,
  splitBatches = FALSE,
```

```

    dryRun = FALSE
)

## S4 method for signature 'featureGroupsSet'
generateFormulasSIRIUS(
  fGroups,
  MSPeakLists,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  relMzDev = defaultLim("mz", "narrow_rel"),
  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.
specSimParams	A named list with parameters that influence the calculation of the annotation similarity . See the spectral similarity parameters documentation for more details.
relMzDev	Maximum relative deviation between the measured and candidate formula <i>m/z</i> 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 patRoön 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 formulae 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.
minIMSSpecSim	(IMS workflow) If the spectrum similarity of an IMS feature group compared to its IMS precursor (see assignMobilities) is at least this value, then the IMS feature group will not be subjected to the annotation algorithm and all feature annotation properties will be copied from its precursor. This assumes that feature annotation is primarily influenced by the MS/MS spectrum, and can be used to speed up the feature annotation process. All scorings, annotation similarities etc. are copied from the IMS precursor. The fragment annotations are also copied (fragInfo result column), however, these are adjusted based on the peak list data of the IMS feature group.
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 `'patRoos.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 spectra into MS peak lists.

Usage

```
generateMSPeakLists(fGroups, ...)

## S4 method for signature 'featureGroups'
generateMSPeakLists(
  fGroups,
  maxMSRTWindow = defaultLim("retention", "narrow"),
  fixedIsolationWidth = FALSE,
  topMost = NULL,
  avgFeatParams = getDefAvgPListParams(),
  avgFGroupParams = getDefAvgPListParams()
)
```

```
## S4 method for signature 'featureGroupsSet'
generateMSPeakLists(fGroups, ...)
```

Arguments

fGroups	The featureGroups object for which MS peak lists should be generated.
...	(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.
fixedIsolationWidth	<p>Configures how MS/MS spectra are selected for a feature:</p> <ul style="list-style-type: none"> • NA: select all spectra unconditionally, <i>i.e.</i> ignoring any precursor information of spectra. • FALSE: select the spectra that were recorded for the feature m/z, using the instrumental precursor isolation window (<i>e.g.</i> quadrupole m/z width) as the selection tolerance. • A numeric value: select the spectra that were recorded for the feature m/z within this tolerance window (+/- precursor m/z). <p>If no isolation was applied to record MS/MS data (<i>e.g.</i> data-independent MS/MS), then all MS/MS spectra will be always be selected.</p>
topMost	Only extract MS peak lists from a maximum of topMost features with highest intensity. If NULL all features 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

Data processing steps that use mass spectral data (*e.g.* [formula generation](#) and [compound generation](#)) typically use 'MS peak lists', which are tables storing the *m/z*, intensity and other data from the raw mass spectra. The generateMSPeakLists function generates MS and MS/MS peak lists first 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. The data processing steps that uses peak lists will either use the 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.

Value

An [MSPeakLists](#) object.

Deprecated algorithms

Prior to **patRoön** 3.0 the `generateMSPeakLists` function was a wrapper to the now deprecated functions `generateMSPeakListsMzR`, `generateMSPeakListsDA` and `generateMSPeakListsDAFMF`. These functions are now unmaintained and may not (fully) work anymore. The current interface is much faster and provides most of the previous functionality (and additional). However, please provide feedback if you feel any functionality is missing.

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by `generateMSPeakLists` to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

The use of profile m/z HRMS data (not IMS-HRMS) is currently not supported.

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, annotation and most other functionality typically uses only the (uncombined and) set specific peak lists, as most algorithms cannot work with mixed peak lists.

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", "ann_form", "ann_comp"
...	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](#), [generateTPsAnnForm](#), [generateTPsAnnComp](#)

The derived classes [transformationProductsFormula](#), [transformationProductsStructure](#), [transformationProductsAnnForm](#) and [transformationProductsAnnComp](#) for more specific methods to post-process TP data.

generateTPsAnnComp	<i>Obtain transformation products (TPs) from compound annotation candidates</i>
--------------------	---

Description

Transforms and prioritizes [compound annotation candidates](#) to obtain TPs.

Usage

```
generateTPsAnnComp(  
  parents,  
  compounds,  
  TPsRef = NULL,  
  fGroupsComps = NULL,  
  minRTDiff = 20,  
  minFitFormula = 0.94,  
  minFitCompound = 0,  
  minSimSusp = 0,  
  minFitCompOrSimSusp = c(0.54, 0.65),  
  extraOptsFMCSR = NULL,  
  skipInvalid = TRUE,  
  prefCalcChemProps = TRUE,  
  neutralChemProps = FALSE,  
  neutralizeTPs = TRUE,  
  TPStructParams = getDefTPStructParams(),  
  parallel = TRUE  
)
```

Arguments

parents The parents for which transformation products should be obtained. This can be

- a suspect list (see [suspect screening](#) for more information)

	<ul style="list-style-type: none">the output of <code>screenSuspects</code> in which case the suspects hits are used as parents <p>The parents need to have SMILES or INCHI information available.</p>
<code>compounds</code>	The <code>compounds</code> object containing the compound candidates.
<code>TPsRef</code>	A <code>transformationProductsStructure</code> object containing suspect TP candidates obtained for the same parents from another TP generation algorithm (<i>e.g.</i> <code>generateTPsBioTransformer</code>). This is used for the calculation of <code>simSusps</code> (see Details). Set to NULL to skip its calculation.
<code>fGroupsComps</code>	The <code>featureGroups</code> object for which the compounds were generated. This is used to obtain retention times for the calculation for <code>retention order directions</code> . Set to NULL to skip its calculation.
<code>minRTDiff</code>	Minimum retention time (in seconds) difference between the parent and a TP to calculate the <code>retention order direction</code> . Candidates with unexpected retention orders are filtered out.
<code>minFitFormula</code> , <code>minFitCompound</code> , <code>minSimSusp</code>	Thresholds to filter out unlikely candidates. For <code>fitFormula</code> : see <code>generateTPsAnnForm</code> , for the others see the Details section.
<code>minFitCompOrSimSusp</code>	A two-sized numeric vector specifying the thresholds for <code>fitCompound</code> <i>or</i> <code>simSusp</code> , respectively.
<code>extraOptsFMCSR</code>	A list with additional options passed to the <code>fmcsR::fmcs</code> function. The following defaults are set: <code>au=1</code> , <code>bu=4</code> , <code>matching.mode="aromatic"</code> .
<code>skipInvalid</code>	If set to TRUE then the parents will be skipped (with a warning) for which insufficient information (<i>e.g.</i> SMILES) is available.
<code>prefCalcChemProps</code>	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 properties section for more details.
<code>neutralChemProps</code>	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 (<i>e.g.</i> <code>[M+H]⁺</code> , <code>[M-H]⁻</code>). See the Validating and calculating chemical properties section for more details.
<code>neutralizeTPs</code>	If TRUE then all resulting TP structure information is neutralized. This argument has a similar meaning as <code>neutralChemProps</code> . This is defaulted to TRUE for prediction algorithms, as these may output charged molecules. NOTE: if neutralization results in duplicate TPs, <i>i.e.</i> when the neutral form of the TP was also generated by the algorithm, then the neutralized TP <i>will be removed</i> .
<code>TPStructParams</code>	Parameters that influence the calculation of structural properties. See <code>getDefTPStructParams</code> .
<code>parallel</code>	If set to TRUE then code is executed in parallel through the future package. Please see the parallelization section in the handbook for more details.

Details

This function uses compound annotations to obtain transformation products. This function is called when calling generateTPs with `algorithm="ann_comp"`.

The generateTPsAnnComp function implements the unknown TP screening from compound candidates approach as described in (Helmus et al. 2025). This algorithm does not rely on any known or predicted TPs and is therefore suitable for 'full non-target' workflows. *All* compound candidates are considered as potential TPs and are ranked by the TP score:

$$TPscore = \max(\text{fitCompound}, \text{simSusp}) + \text{annSim}$$

With:

- annSim: the [annotation similarity](#)
- fitCompound: the structural fit of the compound candidate into the parent (or vice versa, maximum is taken). Calculated as the "Overlap coefficient" with `fmcsR::fmcs`. The molecular data is prepared with `rdk` and `ChemmineR`.
- simSusp: the maximum structural similarity with TP suspect candidates for this parent, *i.e.* obtained from other algorithms of [generateTPs](#). The calculation is configured by the [TP-StructParams](#).

To speed up the calculation process, several thresholds are applied to rule out unlikely candidates. These thresholds are defaulted to those derived in (Helmus et al. 2025). Nevertheless, calculations can take a very long time (multiple hours), especially when processing large numbers of candidates from *e.g.* PubChem.

Unlike most other TP generation algorithms, no additional suspect screening step is required.

Value

generateTPsAnnComp returns an object of the class `transformationProductsAnnComp`. Please see its documentation for *e.g.* filtering steps that can be performed on this object.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formulae 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

Setting `parallel=TRUE` can speed up calculations considerably on multi-core systems. but will also add to RAM usage. Furthermore, parallelization is only favorable for long calculations due to the overhead of setting up multiple R processes. Note that the parallel workers must be on the same system, *i.e.* this will not work on *e.g.* clusters.

It is possible that candidates are equal to their parent. To remove these the `removeParentIsomers` filter can be used afterwards.

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.
- Helmus R, Bagdonaite I, de Voogt P, van Bommel MR, Schymanski EL, van Wezel AP, ter Laak TL (2025). “Comprehensive Mass Spectrometry Workflows to Systematically Elucidate Transformation Processes of Organic Micropollutants: A Case Study on the Photodegradation of Four Pharmaceuticals.” *Environmental Science & Technology*, **59**(7), 3723–3736. ISSN 1520-5851, doi:10.1021/acs.est.4c09121, <http://dx.doi.org/10.1021/acs.est.4c09121>.
- Wang Y, Backman TWH, Horan K, Girke T (2013). “fmcsR: mismatch tolerant maximum common substructure searching in R.” *Bioinformatics*, **29**(21), 2792–2794. ISSN 1367-4811, doi:10.1093/bioinformatics/btt475, <http://dx.doi.org/10.1093/bioinformatics/btt475>.
- Guha R (2007). “Chemical Informatics Functionality in R.” *Journal of Statistical Software*, **18**(6).
- Cao Y, Charisi A, Cheng L, Jiang T, Girke T (2008). “ChemmineR: a compound mining framework for R.” *Bioinformatics*, **24**(15), 1733–1734. ISSN 1367-4803, doi:10.1093/bioinformatics/btn307, <http://dx.doi.org/10.1093/bioinformatics/btn307>.

See Also

[generateTPs](#) for more details and other algorithms.

generateTPsAnnForm	<i>Obtain transformation products (TPs) from formula annotation candidates</i>
--------------------	--

Description

Transforms and prioritizes [formula annotation candidates](#) to obtain TPs.

Usage

```
generateTPsAnnForm(
  parents,
  formulas,
  minFitFormula = 0.94,
  skipInvalid = TRUE,
  prefCalcChemProps = TRUE,
  neutralChemProps = FALSE,
  parallel = TRUE
)
```

Arguments

parents	<p>The parents for which transformation products should be obtained. This can be</p> <ul style="list-style-type: none"> a suspect list (see suspect screening for more information) the output of screenSuspects in which case the suspects hits are used as parents <p>The parents need to have formula information available.</p>
formulas	The formulas object containing the formula candidates.
minFitFormula	Minimum fitFormula (see Details sections) to filter out unlikely candidates.
skipInvalid	If set to TRUE then the parents will be skipped (with a warning) for which insufficient information (<i>e.g.</i> 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 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 (<i>e.g.</i> [M+H] ⁺ , [M-H] ⁻). See the Validating and calculating chemical properties section for more details.
parallel	If set to TRUE then code is executed in parallel through the future package. Please see the parallelization section in the handbook for more details.

Details

This function uses formula annotations to obtain transformation products. This function is called when calling generateTPs with `algorithm="ann_form"`.

The generateTPsAnnForm function implements the unknown TP screening from formula candidates approach as described in (Helmus et al. 2025). This algorithm does not rely on any known or predicted TPs and is therefore suitable for 'full non-target' workflows. All formula candidates are considered as potential TPs and are ranked by the TP score:

$$TPscore = fitFormula + annSim$$

With:

- annSim: the [annotation similarity](#)
- fitFormula: the common element count divided by the total element count for the formulae of the parent/TP or TP/parent (maximum is taken)

To speed up the calculation process, a threshold for fitFormula is applied to rule out unlikely candidates. The default was derived in (Helmus et al. 2025).

Unlike most other TP generation algorithms, no additional suspect screening step is required.

Value

generateTPsAnnForm returns an object of the class [transformationProductsAnnForm](#). Please see its documentation for *e.g.* filtering steps that can be performed on this object.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formulae 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

Setting `parallel=TRUE` may speed up calculations, but is only favorable for long calculations due to the overhead of setting up multiple R processes.

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.

Helmus R, Bagdonaite I, de Voogt P, van Bommel MR, Schymanski EL, van Wezel AP, ter Laak TL (2025). “Comprehensive Mass Spectrometry Workflows to Systematically Elucidate Transformation Processes of Organic Micropollutants: A Case Study on the Photodegradation of Four Pharmaceuticals.” *Environmental Science & Technology*, **59**(7), 3723–3736. ISSN 1520-5851, doi:10.1021/acs.est.4c09121, <http://dx.doi.org/10.1021/acs.est.4c09121>.

See Also

[generateTPs](#) for more details and other algorithms.

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,  
  TPStructParams = getDefTPStructParams(),  
  MP = FALSE  
)
```


Arguments

parents	<p>The parents for which transformation products should be obtained. This can be</p> <ul style="list-style-type: none">• a suspect list (see suspect screening for more information)• the output of screenSuspects in which case the suspects hits are used as parents• a compounds object in which case all candidates are used parents <p>The parents need to have SMILES or INCHI information available.</p>
type	<p>The type of prediction. Valid values are: "env", "ecbased", "cyp450", "phaseII", "hgut", "superbio", "allHuman". Sets the -b command line option.</p>
generations	<p>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.</p>
maxExpGenerations	<p>The maximum number of generations during hierarchy expansion, see below.</p>
extraOpts	<p>A character with extra command line options passed to the biotransformer.jar tool.</p>
skipInvalid	<p>If set to TRUE then the parents will be skipped (with a warning) for which insufficient information (<i>e.g.</i> SMILES) is available.</p>
prefCalcChemProps	<p>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 properties section for more details.</p>
neutralChemProps	<p>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 (<i>e.g.</i> [M+H]⁺, [M-H]⁻). See the Validating and calculating chemical properties section for more details.</p>
neutralizeTPs	<p>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 neutralization results in duplicate TPs, <i>i.e.</i> when the neutral form of the TP was also generated by the algorithm, then the neutralized TP <i>will be removed</i>.</p>
TPStructParams	<p>Parameters that influence the calculation of structural properties. See getDefTPStructParams.</p>
MP	<p>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.</p>

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 [patRoan.path.BioTransformer](#) option. The '.jar' file can be obtained via <https://>

bitbucket.org/djoumbou/biotransformer/src/master. Alternatively, the **patRoönExt** package can be installed to automatically install/configure the necessary files.

Value

The TPs are stored in an object derived from the `transformationProductsStructure` class.

Hierarchy expansion

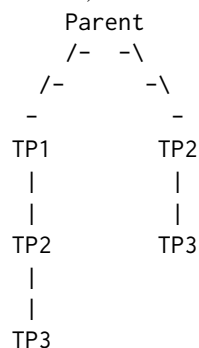
BioTransformer only reports the direct parent for a TP, not the complete pathway. For instance, consider the following results:

- 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, **patRoön** 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 formulae 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 [patRoön 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

- 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.
- Djombou-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, Lorschbach 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	<i>Obtain transformation products (TPs) with Chemical Transformation Simulator (CTS)</i>
----------------	--

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,
  TPStructParams = getDefTPStructParams(),
  parallel = TRUE
)
```

Arguments

parents	<p>The parents for which transformation products should be obtained. This can be</p> <ul style="list-style-type: none"> a suspect list (see suspect screening for more information) the output of screenSuspects in which case the suspects hits are used as parents a compounds object in which case all candidates are used parents <p>The parents need to have SMILES or INCHI information available.</p>
transLibrary	<p>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".</p>
generations	<p>An integer that specifies the number of transformation generations to predict.</p>
errorRetries	<p>The maximum number of connection retries. Sets the times argument to the http::RETRY function.</p>
skipInvalid	<p>If set to TRUE then the parents will be skipped (with a warning) for which insufficient information (e.g. SMILES) is available.</p>
prefCalcChemProps	<p>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 properties section for more details.</p>

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 neutralization 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*.

TPStructParams

Parameters that influence the calculation of structural properties. See [getDefTPStructParams](#).

parallel

If set to TRUE then code is executed in parallel through the **future** 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.

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 formulae 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

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",  
TPStructParams = getDefTPStructParams()  
)
```

Arguments

parents	<p>The parents for which transformation products should be obtained. This can be</p> <ul style="list-style-type: none">• a suspect list (see suspect screening for more information)• the output of screenSuspects in which case the suspects hits are used as parents• a compounds object in which case all candidates are used parents• NULL in which case all parents from the library are used. <p>The parents need to have SMILES or INCHI information available.</p>
TPLibrary	<p>If NULL, a default PubChem based library is used. Otherwise, TPLibrary should be a data.frame. See the details below.</p>
generations	<p>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.</p>
skipInvalid	<p>If set to TRUE then the parents will be skipped (with a warning) for which insufficient information (<i>e.g.</i> SMILES) is available.</p>
prefCalcChemProps	<p>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 properties section for more details.</p>
neutralChemProps	<p>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 (<i>e.g.</i> [M+H]⁺, [M-H]⁻). See the Validating and calculating chemical properties section for more details.</p>
neutralizeTPs	<p>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 neutralization results in duplicate TPs, <i>i.e.</i> when the neutral form of the TP was also generated by the algorithm, then the neutralized TP <i>will be removed</i>.</p>
matchParentsBy	<p>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.</p>

matchGenerationsBy

Similar to matchParentsBy, but specifies how parents/TPs are matched when generations>1.

TPStructParams Parameters that influence the calculation of structural properties. See [getDefTPStructParams](#).

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 is also possible to use your own library.

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 expected [retention order direction](#). (**optional**)
For generateTPsLibrary: If not specified or forceCalcRetDir=TRUE from [TPStructParams](#), then the log P values below may be used to calculate retention order directions.
- 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 formulae 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

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	<p>The parents for which transformation products should be obtained. This can be</p> <ul style="list-style-type: none"> • a suspect list (see suspect screening for more information) • the output of screenSuspects in which case the suspects hits are used as parents <p>The parents need to have formula information available.</p>
TPLibrary	A <code>data.frame</code> . 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 (<i>e.g.</i> 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 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 (<i>e.g.</i> [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 expected [retention order direction](#). (**optional**)
For generateTPsLibrary: If not specified or forceCalcRetDir=TRUE from [TPStructParams](#), then the log P values below may be used to calculate retention order directions.
- 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 formulae 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	<i>Obtain transformation products (TPs) with metabolic logic</i>
------------------	--

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 <code>data.frame</code> with transformation reactions to be used for calculating the TPs (see details below). If NULL, a default table from Schollee <i>et al.</i> is used, that can be obtained with TPLogicTransformations .

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 order direction](#).

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](https://doi.org/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, df = FALSE)
```

```
analysisInfo(obj) <- value  
analyses(obj)  
annotatedPeakList(obj, ...)  
annotations(obj, ...)  
assignMobilities(obj, ...)  
calculatePeakQualities(  
  obj,  
  weights = NULL,  
  flatnessFactor = 0.05,  
  featureQualities = NULL,  
  featureGroupQualities = NULL,  
  ...  
)  
clusterProperties(obj)  
clusters(obj)  
consensus(obj, ...)  
convertToMFDB(TPs, out, ...)  
convertToSuspects(obj, ...)  
cutClusters(obj)  
defaultExclNormScores(obj)  
export(obj, type, out, ...)  
featureTable(obj, ...)  
filter(obj, ...)  
fromIMS(obj)  
getBPCs(obj, ...)  
getFeatures(obj)  
getFeatureQualityNames(obj, ...)
```

```
getMCS(obj, ...)  
getTICs(obj, ...)  
groupNames(obj)  
hasIMS(obj)  
plotBPCs(obj, ...)  
plotChord(obj, addSelfLinks = FALSE, addRetMzPlots = TRUE, ...)  
plotChroms(obj, ...)  
plotChroms3D(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, ...)  
replicates(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.
df	If TRUE then a <code>data.frame</code> is returned, otherwise a <code>data.table</code> is returned.
weights, flatnessFactor, featureQualities, featureGroupQualities	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 <i>m/z</i> 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](#) of an object.

- Methods are defined for: [featureGroups](#); [features](#); [MSPeakListsSet](#).

`analysisInfo<-` modifies the [analysis information](#) of an object.

- Methods are defined for: [featureGroups](#); [featureGroupsXCMS](#); [features](#); [featuresXCMS](#).

`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](#).

`assignMobilities` assigns ion mobility and/or CCS values to workflow data.

- Methods are defined for: [compounds](#); [compoundsSet](#); [featureGroups](#); [featureGroupsScreening](#); [featureGroupsScreeningSet](#); [featureGroupsSet](#).

`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](#); [transformationProductsAnnComp](#); [transformationProductsAnnFo](#); [transformationProductsStructure](#).

`fromIMS` returns TRUE if the object was directly created from IMS data.

- Methods are defined for: [featureGroups](#); [features](#).

`getBPCs` gets base peak chromatogram(s).

- Methods are defined for: [featureGroups](#); [features](#).

`getFeatures` returns the object's [features](#) object.

- Methods are defined for: [featureGroups](#).

`getFeatureQualityNames` returns the object's feature quality names.

- Methods are defined for: [featureGroups](#); [features](#).

`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](#).

`hasIMS` returns TRUE if the object has ion mobility values

- Methods are defined for: [featureGroups](#); [featureGroupsComparison](#); [features](#).

`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](#).

`plotChroms3D` plots a three dimensional chromatogram.

- Methods are defined for: [featureGroups](#); [featureGroupsSet](#).

`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](#).

`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](#); [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](#); [featuresPiek](#); [featuresXCMS](#); [featuresXCMS3](#); [formulas](#); [formulasSet](#); [formulasSIRIUS](#); [MSLibrary](#); [MSPeakLists](#); [MSPeakListsSet](#); [transformationProducts](#).

`plotVolcano` plots a volcano plot.

- Methods are defined for: [featureGroups](#).

`replicates` returns a character vector with the replicates 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`; `featureGroupsScreening`, `ANY`, `ANY`, `missing`; `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](#); [C++Object](#); [components](#); [componentsFeatures](#); [componentsSet](#); [compounds](#); [compoundsCluster](#); [compoundsSet](#); [featureGroups](#); [featureGroupsScreening](#); [featureGroupsScreeningSet](#); [featureGroupsSet](#); [features](#); [featuresSet](#); [formulas](#); [formulasSet](#); [MSLibrary](#); [MSPeakLists](#); [MSPeakListsSet](#); [optimizationResult](#); [transformationProducts](#); [workflowStep](#); [workflowStepSet](#).

genFormulaTPLibrary	<i>Automatically generate a transformation product library with formula data.</i>
---------------------	---

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 <i>et al.</i> is used, that can be obtained with TPLogicTransformations .
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 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 (<i>e.g.</i> [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 order direction](#).

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 formulae 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](#)

getBGMSMSPeaks

Background MS/MS peak detection

Description

Detects background MS/MS peaks by gathering frequently occurring peaks in MS/MS spectra from blanks.

Usage

```
getBGMSMSPeaks(  
  anaInfo,  
  replicates = NULL,  
  MSLevel = 2,  
  retentionRange = NULL,  
  mobilityRange = NULL,  
  minBPIntensity = 5000,  
  avgSpectraParams = getDefAvgPListParams(relMinAbundance = 0.1, topMost = 25),  
  avgAnalysesParams = getDefAvgPListParams(relMinAbundance = 0.8, topMost = 25)  
)
```


Arguments

anaInfo	The analysis info object with the analyses to be used for background peak detection.
replicates	A character with names of replicates for the analyses used for background peak detection. If NULL then all the analyses defined in anaInfo will be used.
MSLevel	The MS level of the spectra to be used for background peak detection. This should be '1' or '2'. This function is only tested with MSLevel=2. And while MSLevel=1 is possible, it may be rather computationally intensive due to the relatively large number of MS peaks to iterate through.
retentionRange, mobilityRange	A two-sized numeric vector with the minimum and maximum retention time and ion mobility to consider, respectively. If NULL then the full range is used.
minBPIntensity	The minimum basepeak intensity of a spectrum to be considered for background peak detection. This is primarily intended to optimize the detection procedure.
avgSpectraParams, avgAnalysesParams	A list with parameters used for averaging the MS spectra within and between analyses, respectively. See getDefAvgPListParams for more details.

Details

This function iterates through all MS/MS spectra of the given analyses and collects the most frequently occurring peaks. It first averages all spectra within the same analyses, and retains those peaks above an abundance threshold (set by avgSpectraParams). The analyses-averaged spectra are then also averaged, and again peaks with a minimum abundance are kept (set by avgAnalysesParams).

The frequent occurrence of a peak throughout MS/MS spectra, including DDA spectra different isolation m/z values, is often a sign of contamination from the analytical system (*e.g.* from the mobile phase, MS quadrupoles etc). Hence, the analyses used for blank subtraction are typically just measurements of *e.g.* ultrapure water or another solvent, and not need to be treated by any extraction procedure.

The output of this function can directly be used to the [filter method for MSPeakLists](#) to subsequently remove these background peaks, which possibly improves subsequent feature annotation.

Value

A [data.table](#) with the detected background peaks and abundance statistics. The table can directly be passed to the removeMZs argument of the [filter method for MSPeakLists](#).

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by getBGMSMSPeaks to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

The use of profile m/z HRMS data (not IMS-HRMS) is currently not supported.

References

Helmus R, Bagdonaite I, de Voogt P, van Bommel MR, Schymanski EL, van Wezel AP, ter Laak TL (2025). “Comprehensive Mass Spectrometry Workflows to Systematically Elucidate Transformation Processes of Organic Micropollutants: A Case Study on the Photodegradation of Four Pharmaceuticals.” *Environmental Science & Technology*, **59**(7), 3723–3736. ISSN 1520-5851, doi:10.1021/acs.est.4c09121, <http://dx.doi.org/10.1021/acs.est.4c09121>.

getCCSParams

Parameters for CCS calculation

Description

Configuration and description of parameters used for CCS calculation.

Usage

```
getCCSParams(method, ..., calibrant = NULL)
```

Arguments

method, calibrant

Sets the CCS calculation method and the calibrant data (only required if method="agilent"). See details.

...

optional named arguments that override defaults.

Details

The following parameters exist to configure the CCS calculation:

- **method** The CCS calculation method. Should be "bruker", "mason-schamp_k", "mason-schamp_1/k" or "agilent". See details below.
- **defaultCharge** The default charge of the ions. This is used when no charge information is available.
- **temperature,massGas** The temperature (Kelvin) and exact mass of the drift gas. See calculation details below.
- **MasonSchampConstant** The Mason-Schamp constant. See calculation details below.
- **calibrant** If method="agilent": the calibrant data to be used for CCS calculation. This should either be
 - A path to an Agilent '.d' file.
 - A path to an 'OverrideImsCal.xml' file (found in 'sample.d/AcqData').
 - A named list with the elements massGas, TFix and beta.

The CCS calculation depends on the method parameter:

- **bruker**: uses the Bruker TDF-SDK for calculations. See [msdata](#) for configuration options. Only applicable to TIMS data.

- mason-schamp_k: uses the Mason-Schamp equation:

$$CCS = C \cdot \frac{charge}{\sqrt{u \cdot T}} \cdot \frac{1}{mobility}$$

With

- C the Mason-Schamp constant, can be changed by setting the `MasonSchampConstant` parameter. See (George et al. 2024) for details.
- u the reduced mass of the drift gas and the ion:

$$u = \frac{m_{gas} \cdot m_{ion}}{m_{gas} + m_{ion}}$$

The mass of the drift gas is defined by the `massGas` parameter.

- T the temperature (Kelvin) as defined by the `temperature` parameter.
- mason-schamp_1/k: as mason-schamp_k but assuming an inversed mobility ($\frac{1}{k}$). This is meant for TIMS data. Compared to `method="bruker"`, this doesn't rely on the TDF-SDK but may produce results with very minor differences (George et al. 2024).
- agilent: uses Agilent calibration data with the following equation:

$$CCS = (mobility - t_{fix}) \cdot \frac{charge}{\beta} \cdot \frac{1}{\sqrt{\frac{m_{ion}}{m_{ion} + m_{gas}}}}$$

With t_{fix} and β the TFix and beta values from the calibration data. The `massGas` parameter sets the m_{gas} value.

The `getCCSParams` function generates such parameter list with defaults.

Source

The calculation formulas was derived from (Haler et al. 2017), (George et al. 2024) and the implementation used by **MS-DIAL** (Tsugawa et al. 2020). (`MobilityToCrossSection` method from the `IonMobilityUtility` class).

References

- George AC, Schmitz I, Rouviere F, Alves S, Colsch B, Heinisch S, Afonso C, Fenaille F, Loutelier-Bourhis C (2024). "Interplatform comparison between three ion mobility techniques for human plasma lipid collision cross sections." *Analytica Chimica Acta*, **1304**, 342535. ISSN 0003-2670, doi:10.1016/j.aca.2024.342535, <http://dx.doi.org/10.1016/j.aca.2024.342535>.
- Haler JRN, Kune C, Massonnet P, Comby-Zerbino C, Jordens J, Honing M, Mengerink Y, Far J, De Pauw E (2017). "Comprehensive Ion Mobility Calibration: Poly(ethylene oxide) Polymer Calibrants and General Strategies." *Analytical Chemistry*, **89**(22), 12076–12086. ISSN 1520-6882, doi:10.1021/acs.analchem.7b02564, <http://dx.doi.org/10.1021/acs.analchem.7b02564>.
- Tsugawa H, Ikeda K, Takahashi M, Satoh A, Mori Y, Uchino H, Okahashi N, Yamada Y, Tada I, Bonini P, Higashi Y, Okazaki Y, Zhou Z, Zhu Z, Koelmel J, Cajka T, Fiehn O, Saito K, Arita M, Arita M (2020). "A lipidome atlas in MS-DIAL 4." *Nature Biotechnology*, **38**(10), 1159–1163. ISSN 1546-1696, doi:10.1038/s4158702005312, <http://dx.doi.org/10.1038/s41587-020-0531-2>.

getDefAvgPListParams *Parameters for averaging MS peak list data*

Description

Create parameter lists for averaging MS peak list data.

Usage

```
getDefAvgPListParams(..., IMS = getLimIMS())
```

Arguments

...	Optional named arguments that override defaults.
IMS	A character that specifies for which IMS instrument defaults are returned. Should be "bruker" or "agilent". Defaults to what is specified in limits .

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:

- `method,clusterMzWindow` The cluster method and window (see [clustering parameters](#)) used to average mass spectra. `clusterMzWindow` is defaulted as `defaultLim("mz", "medium")` (see [limits](#)).
- `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` and averaging).
- `minIntensityPost` MS peaks with intensities below this value will be removed (after averaging).
- `minIntensityIMS` MS peaks in spectra of raw IMS frames with intensities below this value will be removed (applied prior to any other treatment steps).
- `absMinAbundance,relMinAbundance` Minimum absolute/relative abundance of an MS peak across the spectra that are averaged. If `absMinAbundance` exceeds the number of spectra then the threshold is automatically lowered to the number of spectra.
- `minRelCumIntensity` Minimum relative cumulative intensity of an MS peak in the averaged spectrum.
- `smoothWindowIMS, halfWindowIMS, maxGapIMS` Parameters used for centroiding m/z peaks from IMS-HRMS data. See [Centroiding IMS data](#) for more details.

- **withPrecursorMS** For MS data only: ignore any spectra that do not contain the precursor peak.
For IMS data this excludes MS spectra within an IMS frame that do not contain the precursor peak, typically due to mobility separation. Hence, setting this option performs some crude cleanup of MS spectra, even for features for which no mobilities were assigned (*e.g.* non-IMS workflows).
- **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 calculation) 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 defaults are (with `IMS="bruker"`):

```
list(
  method = "distance_mean",
  clusterMzWindow = 0.005,
  topMost = 50,
  minIntensityPre = 500,
  minIntensityPost = 500,
  minIntensityIMS = 25,
  absMinAbundance = 0,
  relMinAbundance = 0,
  minRelCumIntensity = 0,
  smoothWindowIMS = 0,
  halfWindowIMS = 2,
  maxGapIMS = 0.005,
  withPrecursorMS = TRUE,
  pruneMissingPrecursorMS = TRUE,
  retainPrecursorMSMS = TRUE
)
```

Value

`getDefAvgPListParams` returns a list with the peak list averaging parameters.

Centroiding IMS data

With IMS-HRMS data the m/z peaks are often not or partially centroided. The following steps are performed to centroid the data:

1. Sum up mass spectra within an IMS frame. If the feature has mobility data, only spectra within its mobility boundaries are considered.

2. Use point-distance clustering (see [clustering parameters](#)) with a window defined by maxGapIMS to find related mass signals. This is primarily meant for non-continuous data, *e.g.* due to intensity thresholding. The maxGapIMS parameter should be set to a value that represents the maximum expected distance between two m/z datapoints. For some instruments, such as Agilent IMS-QTOF, this value may be higher than expected. For that reason, if IMS="agilent" then the default is set to 0.01.
3. Smooth the intensity data using a centered moving average with window size smoothWindowIMS (set to zero to disable smoothing).
4. Find local maxima within sliding window with +/- halfWindowIMS points and eliminate non-centroids. This algorithm is based on the C_localMaxima function from **MALDIquant**.

References

Gibb S, Strimmer K (2012). "MALDIquant: a versatile R package for the analysis of mass spectrometry data." *Bioinformatics*, **28**(17), 2270–2271. doi:10.1093/bioinformatics/bts447.

getDefPeakParams	<i>Peak detection parameters</i>
------------------	----------------------------------

Description

Algorithms and parameters for automatic detection of peaks in chromatograms and mobilograms.

Usage

```
getDefPeakParams(type, algorithm, ...)
```

Arguments

type	The type of parameter defaults: "chrom" for chromatograms and "bruker_ims" and "agilent_ims" for mobilograms coming from Bruker and Agilent systems, respectively.
algorithm	The peak detection algorithm: "openms", "xcms3", "envipick" or "piek".
...	optional named arguments that override defaults.

Details

The algorithm and its parameters for peak detection should be in a named list with the format:

```
list(algorithm = <algorithm>, param1 = ..., param2 = ..., ...)
```

Where <algorithm> is the name of the algorithm and param1, param2 etc are the parameters. The getDefPeakParams function generates such parameter list with the algorithm and default parameters.

The following algorithms are currently supported:

- "openms": uses **MRMTransitionGroupPicker** tool from **OpenMS**.

- "xcms3": uses the `xcms::peaksWithCentWave` function.
- "envipick": uses the `enviPick::mzpick` function.
- "piek": uses the peak detection algorithm from (Dietrich et al. 2021), which was optimized with **OpenMP** parallelization. See `findFeaturesPiek` for more details.

The parameters are discussed in the next sections.

General parameters

These parameters are applicable to all algorithms

- `forcePeakWidth` a two-sized numeric vector with the minimum and maximum width for a peak. Peaks that are more narrow or wide will be clamped to this range. This is especially useful for algorithms that consider an extensive part of the fronting/tailing noise as part as the peak. Set to `c(0, 0)` to disable.
- `relMinIntensity` the minimum intensity threshold for a peak relative to the highest peak in the same chromatogram/mobilogram. This is *e.g.* useful to exclude noise in mobilograms where normally few peaks are expected.
- `calcCentroid` Controls how the peak centroid is calculated, which is used for retention time or mobility determination. Valid values are: "algorithm" (use the centroid as determined by the algorithm), "max" (use the apex of the peak), "weighted.mean" (use the intensity weighted mean of all data points in the peak) or "centerOfMass" (use the center of mass or first statistical moment of the peak). The latter two might of interest for assymetrical peaks. However, most algorithms, including those not interfaced by **patRoan**, seem to use the peak apex. Hence, `calcCentroid="max"` (or `calcCentroid="algorithm"` which is usually the same) seems a good default for comparative reasons.

Parameters for openms

The parameters directly map to the command line options for `MRMTransitionGroupPicker`, please see [its documentation](#).

- `minPeakWidth` the minimum peak width, sets the `min_peak_width` option.
- `backgroundSubtraction` the background subtraction method, sets the `-algorithm:background_subtraction` option.
- `SGolayFrameLength` the frame length for Savitzky-Golay smoothing, sets the `-algorithm:PeakPickerMRM:sgolay_frame_length` option.
- `SGolayPolyOrder` order of the polynomial, sets the `-algorithm:PeakPickerMRM:sgolay_polynomial_order` option.
- `useGauss` set to TRUE to use Gaussian smoothing (instead of Savitzky-Golay, sets the `-algorithm:PeakPickerMRM:use_gauss` option.
- `gauss_width` the Gaussian width, estimated peak size, sets the `-algorithm:PeakPickerMRM:gauss_width` option.
- `SN` signal to noise threshold, sets the `-algorithm:PeakPickerMRM:signal_to_noise` option.
- `SNWinLen` SN window length, sets the `-algorithm:PeakPickerMRM:sn_win_len` option.
- `SNBinCount` SN bin count, sets the `-algorithm:PeakPickerMRM:sn_bin_count` option.

- method peak picking method, sets the `-algorithm:PeakPickerMRM:method` option.
- integrationType the integration technique, sets the `-algorithm:PeakIntegrator:integration_type` option.
- baselineType the baseline type, sets the `-algorithm:PeakIntegrator:baseline_type` option.
- fitEMG if TRUE then the EMG model is used for fitting, sets the `-algorithm:PeakIntegrator:fit_EMG` option.

Parameters for xcms3 and envipick

See the documentation for `xcms::peaksWithCentWave` and `enviPick::mzpick` for xcms3 and envipick, respectively.

Parameters for piek

- minIntensity the minimum intensity of a peak.
- SN the signal to noise ratio.
- peakWidth two-sized vector with the minimum and maximum peak width (seconds)
- RTRange two-sized vector with the minimum and maximum retention time range (seconds). Set the 2nd element to Inf for no upper limit.
- maxPeaksPerSignal upper threshold for consecutive maxima of similar size to be regarded as noise.

Note

The peak detection used by `algorithm="openms"` is different than that of `findFeaturesOpenMS`. The `patRoon.threads` package option sets the number of threads for the piek algorithm.

References

Rost HL, Sachsenberg T, Aiche S, Bielow C, Weissner 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 (2026). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.1, <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.

Dietrich C, Wick A, Ternes TA (2021). “Open-source feature detection for non-target LC–MS analytics.” *Rapid Communications in Mass Spectrometry*, **36**(2). ISSN 1097-0231, doi:10.1002/rcm.9206, <http://dx.doi.org/10.1002/rcm.9206>. 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.

getDefTPStructParams *Parameters to handle TP data with structural information*

Description

Parameters used by `generateTPs` with algorithms that use structural information.

Usage

```
getDefTPStructParams(...)
```

Arguments

... optional named arguments that override defaults.

Details

The following parameters can be configured:

- `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 are values of parents and TPs are used for [retention order calculation](#).
- `forceCalcLogP` Force calculation of Log P values, even if already provided by the TP generation algorithm. This is primarily useful to obtain log P values that were consistently calculated with the same algorithm, as some algorithms may only partially output these values (*e.g.* not for parents).
- `forceCalcRetDir` Force calculation of [retention order directions](#), even if already provided by the TP generation algorithm. This is primarily intended for re-calculation of library TP data, which may have been calculated with different log P values.
- `minLogPDiff` The minimum difference in log P values between a parent and its TPs to be considered eluting differently. This is used for [retention order calculation](#).

- `calcSims` If set to TRUE then structural similarities between the parent and its TPs are calculated. The `filter method` can be used to threshold structural similarities. This 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 `rdk`.
- `fpSimMethod` The method for calculating similarities (i.e. not dissimilarity!). See the method argument of the `fp.sim.matrix` function of the `fingerprint` package.

These parameters are passed as a named list as the `TPStructParams` argument to functions.

The `getDefTPStructParams` function generates such parameter list with defaults.

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.

getEICs

Obtains extracted ion chromatograms (EICs)

Description

This function generates one or more EIC(s) for given retention time, m/z and optionally mobility ranges.

Usage

```
getEICs(
  analysisInfo,
  ranges,
  gapFactor = 3,
  output = "fill",
  minIntensityIMS = 25
)
```

Arguments

<code>analysisInfo</code>	A data.frame (or data.table) with Analysis information .
<code>ranges</code>	A list with for each analysis a data.frame with numeric columns "retmin", "retmax", "mzmin", "mzmax" with the lower/upper ranges of the retention time and m/z . Furthermore, columns "mobmin" and "mobmax" can be added for mobility lower/upper ranges in IMS data.
<code>gapFactor</code>	A numeric that configures gap filling. See getDefEICParams for more details.

output Should be "fill", "pad" or "raw". Internally, EIC data is compressed by omitting any zero intensity data points. If output="fill" then the zero intensity points are re-added to obtain continuous chromatograms. If output="pad" then zero intensity points are only re-added that surround others, which is sufficient for *e.g.* plotting. If output="raw" then the original compressed data is returned.

minIntensityIMS (**IMS workflow**) Raw intensity threshold for IMS data. This is primarily intended to speed up raw data processing.

Value

A list with for each analysis a list with EIC data for each of the rows in ranges.

If output="raw" then additional columns with *e.g.* mean-averaged and base peak m/z values for each data point are returned. Furthermore, the allXValues attribute is set that can be used to obtain the original retention time values to reconstruct the original complete chromatogram.

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by getEICs to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

getFCParams	<i>Fold change calculation</i>
-------------	--------------------------------

Description

Fold change calculation

Usage

```
getFCParams(replicates, ...)
```

Arguments

replicates A character vector with the names of the two replicates to be compared.

... Optional named arguments that override defaults.

Details

Fold change calculation can be used to easily identify significant changes between replicates. The calculation process is configured through a parameter list, which can be constructed with the getFCParams function. The parameter list has the following entries:

- replicates the name of the two replicates to compare (taken from the replicates 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](#)

getIMSMatchParams	<i>Parameters for IMS matching</i>
-------------------	------------------------------------

Description

Parameters that define how mobility or CCS values between *e.g.* features and suspects should be matched.

Usage

```
getIMSMatchParams(param, ...)
```

Arguments

param	Should be "mobility" or "CCS" to match by mobility or CCS, respectively.
...	optional named arguments that override defaults.

Details

The following parameters should be defined:

- param Should be "mobility" or "CCS" to match by mobility or CCS, respectively.
- window,relative The window parameter sets the tolerance window size used for matching. If relative=TRUE then the tolerance is relative ('0-1'). The defaults for window are (see [limits](#)):
 - defaultLim("mobility", "medium") (param="mobility" and relative=FALSE)
 - defaultLim("mobility", "medium_rel") (param="mobility" and relative=TRUE)
 - defaultLim("CCS", "medium") (param="CCS" and relative=FALSE)
 - defaultLim("CCS", "medium_rel") (param="CCS" and relative=TRUE)
- minMatches The minimum number of mobility/CCS matches for a suspect hit. If the number of available reference mobility/CCS values in the suspect list is less than minMatches, then that number is used as threshold. Set to 0 to disable.

These parameters are passed as a named list as the IMSMatchParams argument to functions.

The getIMSMatchParams function generates such parameter list with defaults.

Note

If negation is enabled with suspect filtering and minMatches>0, then the window match filter is *not* negated. Negating both would lead to unexpected results, *i.e.* suspects outside window are kept and increase the number of matched suspects as seen by the minMatches filter.

getMSRangeParams	<i>Parameters to specify a IMS data range</i>
------------------	---

Description

Parameters that define a range of mobility or CCS values.

Usage

```
getMSRangeParams(param, lower, upper, mzRelative = FALSE)
```

Arguments

param, lower, upper, mzRelative

Arguments to specify the IMS range parameters, see Details.

Details

The following parameters are used to define an IMS range:

- param Should be "mobility" or "CCS" to specify a mobility or CCS range, respectively.
- lower,upper The lower and upper range.
- mzRelative Set to TRUE to specify an IMS range that is normalized by m/z .

These parameters are passed as a named list as the MSRangeParams argument to functions.

The getMSRangeParams function generates such parameter list with defaults.

getMSFileFormats	<i>Get supported MS file formats</i>
------------------	--------------------------------------

Description

Returns the supported file formats of MS files in **patRoön**.

Usage

```
getMSFileFormats(fileType = NULL)
```

Arguments

fileType The type of file for which formats should be returned (see [getMSFileTypes](#)). If NULL then all file formats are returned.

getMSFileTypes	<i>Get supported MS file types</i>
----------------	------------------------------------

Description

Get supported MS file types

Usage

```
getMSFileTypes()
```

groupFeatures	<i>Grouping of features</i>
---------------	-----------------------------

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 <code>data.frame</code> with analysis info to be passed to <code>groupFeaturesSIRIUS</code>
algorithm	A character that specifies the algorithm to be used: either "openms", "xcms", "xcms3", "kpic2" or "greedy" (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](#), [groupFeaturesGreedy](#), [groupFeaturesSIRIUS](#)

groupFeaturesGreedy	<i>Group features using greedy algorithm</i>
---------------------	--

Description

Group features using a greedy algorithm that maximizes group scores based on retention time, m/z, mobility, and intensity similarities.

Usage

```
groupFeaturesGreedy(feat, ...)

## S4 method for signature 'features'
groupFeaturesGreedy(
  feat,
  rtalign = FALSE,
  rtWindow = defaultLim("retention", "medium"),
  mzWindow = defaultLim("mz", "medium"),
  mobWindow = defaultLim("mobility", "medium"),
  scoreWeights = c(retention = 1, mz = 1, mobility = 1, intensity = 1),
  verbose = TRUE
)
```

Arguments

feat	The features object with the features to be grouped.
...	Further parameters passed to the selected grouping algorithm.
rtalign	Not yet supported. Provided for consistency with other grouping methods.
rtWindow, mzWindow, mobWindow	Numeric tolerances for retention time (seconds), m/z , and mobility, respectively. The scoring terms are normalized to these values. Defaults to <code>defaultLim("retention", "medium")</code> , <code>defaultLim("mz", "medium")</code> , and <code>defaultLim("mobility", "medium")</code> , respectively (see limits).
scoreWeights	Numeric vector specifying the scoring weights. Should contain the following named elements: "retention", "mz", "mobility", and "intensity". Missing elements are defaulted to one.
verbose	if FALSE then no text output will be shown.

Details

This function uses greedy to group features. This function is called when calling `groupFeatures` with `algorithm="greedy"`.

The greedy algorithm is a simple feature grouping algorithm that can work with both HRMS and IMS-HRMS data. The algorithm groups features by iteratively building the best possible groups. Features are processed in order of decreasing intensity. For each feature, candidate groups are formed from all other (ungrouped) features within the specified retention time, m/z and mobility windows. Each candidate group only contains a maximum of one feature per analysis. The candidates are then scored and the group with the lowest overall variations in retention time, m/z , mobility and replicate intensity is then selected. This process is repeated until all features have been assigned to a group. The weights for each of the scoring terms can be configured.

Value

An object of a class which is derived from [featureGroups](#).

Note

Any links between IMS precursors and IMS features are removed. This can occur *e.g.* when greedy is used to generate a [feature consensus](#) from a [post mobility assignment](#) workflow.

See Also

[groupFeatures](#) for more details and other algorithms.

groupFeaturesKPIC2	Group features using KPIC2
--------------------	----------------------------

Description

Uses the the **KPIC2** R package for grouping of features.

Usage

```
groupFeaturesKPIC2(feats, ...)  
  
## 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 (<i>e.g.</i> 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(feats, ...)  
  
## S4 method for signature 'features'  
groupFeaturesOpenMS(  
  feats,  
  rtalign = TRUE,  
  QT = FALSE,  
  maxAlignRT = defaultLim("retention", "wide"),  
  maxAlignMZ = defaultLim("mz", "medium"),  
  maxGroupRT = defaultLim("retention", "medium"),  
  maxGroupMZ = defaultLim("mz", "medium"),  
  extraOptsRT = NULL,  
  extraOptsGroup = NULL,  
  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 <code>-algorithm:pairfinder:distance_RT:max_difference</code> and <code>-algorithm:pairfinder:distance_MZ:max_difference</code> options, respectively.
maxGroupRT, maxGroupMZ	as maxAlignRT and maxAlignMZ, but for grouping of features. Sets <code>-algorithm:distance_RT:max_difference</code> and <code>-algorithm:distance_MZ:max_difference</code> 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: <code>extraOptsGroup=list("-algorithm:d")</code> (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 (2026). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.1, <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.

See Also

[groupFeatures](#) for more details and other algorithms.

groupFeaturesSIRIUS	<i>Group features using SIRIUS</i>
---------------------	------------------------------------

Description

Uses **SIRIUS** to find *and* group features.

Usage

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

Arguments

analysisInfo	A data.frame (or data.table) 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	<i>Group features using XCMS (old interface)</i>
-------------------	--

Description

Group and align features with the legacy [xcmsSet](#) function from the **xcms** package.

Usage

```
groupFeaturesXCMS(feats, ...)  
  
## S4 method for signature 'features'  
groupFeaturesXCMS(  
  feats,  
  rtalign = TRUE,  
  loadRawData = TRUE,  
  groupArgs = list(mzwid = 0.015),  
  retcorArgs = list(method = "obiwarp"),  
  verbose = TRUE  
)  
  
## S4 method for signature 'featuresSet'  
groupFeaturesXCMS(feats, groupArgs = list(mzwid = 0.015), verbose = TRUE)
```

Arguments

<code>feats</code>	The features object with the features to be grouped.
<code>...</code>	Further parameters passed to the selected grouping algorithm.
<code>rtalign</code>	Set to TRUE to enable retention time alignment.
<code>loadRawData</code>	Set to TRUE if analyses are available as mzXML or mzML files. Otherwise MS data is not loaded, and some dummy data (<i>e.g.</i> file paths) is used in the returned object.
<code>groupArgs</code>	named character vector that may contain extra grouping parameters to be used by xcms::group
<code>retcorArgs</code>	named character vector that may contain extra parameters to be used by xcms::retcor .
<code>verbose</code>	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	<i>Group features using XCMS (new interface)</i>
--------------------	--

Description

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

Usage

```
groupFeaturesXCMS3(feats, ...)  
  
## S4 method for signature 'features'  
groupFeaturesXCMS3(  
  feats,  
  rtalign = TRUE,
```

```

    loadRawData = TRUE,
    groupParam = xcms::PeakDensityParam(sampleGroups = analysisInfo(feats)$replicate),
    preGroupParam = groupParam,
    retAlignParam = xcms::ObiwrapParam(),
    verbose = TRUE
)

## S4 method for signature 'featuresSet'
groupFeaturesXCMS3(
  feat,
  groupParam = xcms::PeakDensityParam(sampleGroups = analysisInfo(feats)$replicate),
  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 (<i>e.g.</i> 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 <i>prior</i> 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.

id-conf

Identification confidence estimation

Description

Functions to estimate the identification confidence for suspects and annotation candidates.

Usage

```
estimateIDConfidence(obj, ...)

## S4 method for signature 'formulas'
estimateIDConfidence(
  obj,
  absMzDev = defaultLim("mz", "medium"),
  normalizeScores = "max",
  IDFile = system.file("misc", "IDLevelRules.yml", package = "patRoan"),
  logPath = NULL
)

## S4 method for signature 'compounds'
estimateIDConfidence(
  obj,
  absMzDev = defaultLim("mz", "medium"),
  MSPeakLists = NULL,
  formulas = NULL,
  specSimParams = getDefSpecSimParams(removePrecursor = TRUE),
  formulasNormalizeScores = "max",
  compoundsNormalizeScores = "max",
  IDFile = system.file("misc", "IDLevelRules.yml", package = "patRoan"),
  logPath = NULL
)
```



```
## S4 method for signature 'featureGroupsScreening'
estimateIDConfidence(
  obj,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  absMzDev = defaultLim("mz", "medium"),
  checkFragments = c("mz", "formula", "compound"),
  formulasNormalizeScores = "max",
  compoundsNormalizeScores = "max",
  IDFile = system.file("misc", "IDLevelRules.yml", package = "patRoan"),
  logPath = file.path("log", "ident")
)

## S4 method for signature 'featureGroupsScreeningSet'
estimateIDConfidence(
  obj,
  MSPeakLists = NULL,
  formulas = NULL,
  compounds = NULL,
  absMzDev = defaultLim("mz", "medium"),
  checkFragments = c("mz", "formula", "compound"),
  formulasNormalizeScores = "max",
  compoundsNormalizeScores = "max",
  IDFile = system.file("misc", "IDLevelRules.yml", package = "patRoan"),
  logPath = file.path("log", "ident")
)

## S4 method for signature 'compoundsSet'
estimateIDConfidence(
  obj,
  absMzDev = defaultLim("mz", "medium"),
  MSPeakLists = NULL,
  formulas = NULL,
  formulasNormalizeScores = "max",
  compoundsNormalizeScores = "max",
  IDFile = system.file("misc", "IDLevelRules.yml", package = "patRoan"),
  logPath = NULL
)

## S4 method for signature 'formulasSet'
estimateIDConfidence(
  obj,
  absMzDev = defaultLim("mz", "medium"),
  normalizeScores = "max",
  IDFile = system.file("misc", "IDLevelRules.yml", package = "patRoan"),
  logPath = NULL
)
```

```
)

numericIDLevel(level)

genIDLevelRulesFile(out, inLevels = NULL, exLevels = NULL)
```

Arguments

obj	The object for which identification confidence should be estimated.
...	Method specific arguments.
absMzDev	Maximum absolute m/z deviation.
normalizeScores, 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 <code>Suspect</code> 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. NOTE: To avoid slowdowns by logging for potentially large number of candidates, logging is disabled for the <code>formulas</code> and <code>compounds</code> methods by default.
MSPeakLists, formulas, compounds	Annotation data (<code>MSPeakLists</code> , <code>formulas</code> and <code>compounds</code>). All arguments can be NULL, but it is recommended to set them if possible to allow the most complete estimations.
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 <code>fragments_mz/fragments_formula</code> columns in the suspect list). Valid values are: "mz", "formula", "compounds". The former uses m/z values in the specified <code>MSPeakLists</code> object, whereas the others use the formulae that were annotated to MS/MS peaks in the given <code>formulas</code> or <code>compounds</code> objects. Multiple values are possible: in this case the maximum number of fragment matches will be reported.
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, <code>exLevels="4 5"</code> would exclude level 4 and 5 from the output file. Set to NULL to ignore.

Details

The `estimateIDConfidence` methods are used to estimate various properties to estimate the confidence of identifications assigned to suspects and feature annotation candidates. These functions are typically executed after running `screenSuspects`, `generateFormulas` and `generateCompounds`. Afterwards, the following columns are added to the result tables (obtained with *e.g.* `screenInfo`, `annotations` and `as.data.table`):

- `annSim` The *annotation similarity*, defined as the similarity between the MS/MS peak list of a feature with (a) only the peaks that were annotated and (b) all the peaks. Thus, a value of one means that all MS/MS peaks were annotated. The similarity calculation is configured with the `specSimParams` argument to `estimateIDConfidence`.
- `annSimForm` The annotation similarity specifically for formula annotations (equaling the `annSim` column from formula annotations). Only calculated for `suspects` and `compounds`.
- `annSimBoth` The annotation similarity calculated with the combined set of annotated MS/MS peaks from formula and compound annotations. Only calculated for `suspects` and `compounds`.
- `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.

In addition, the following columns are specifically added to suspect screening results:

- `annSimComp` The annotation similarity specifically for compound annotations (this equals the `annSim` column in compound annotations).
- `formRank, compRank` The rank of the suspect within the formula/compound annotation results.
- `maxFrag` 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 `maxFrag`. The calculation of this column is influenced by the `checkFragments` argument to `estimateIDConfidence`.

The data for these columns is only calculated if `estimateIDConfidence` has the required data to do so. For instance, `annSimForm` and `formRank` are only calculated if the `formulas` argument is set, and levels for `estIDLevel` will be poor if no compound annotations are available.

`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 `estimateIDConfidence`.

Value

`estimateIDConfidence` amends the input object with aforementioned identification confidence properties.

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: 0.7
    - libMatch: 0.7
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 (maxFrag 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.
- Any conditions that require suspect data (e.g. suspectFragments) are only met with the suspects method for estimateIDConfidence method.

A template rules file can be generated with the `genIDLevelRulesFile` function, and this file can subsequently be passed to `estimateIDConfidence`. 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.

Sets workflows

`estimateIDConfidence` performs its estimations per set. In addition, the following overall (not set specific) columns are calculated:

- formRank and compRank based on the ranking of the formula/compound in the set consensus data.
- estIDLevel: 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').
- Annotation similarities: taken as the maximum value from the data for each set.

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.

importFeatureGroups	<i>Import feature groups from files</i>
---------------------	---

Description

Generic function to import feature groups produced by other software from files.

Usage

```
importFeatureGroups(input, type, ...)
```

Arguments

input	The input object or path that should be imported. See the algorithm specific functions for more details.
type	What type of data should be imported: "xcms", "xcms3", "kpic2", "table", "brukerpa" (Bruker ProfileAnalysis), "brukertasq" (Bruker TASQ) or "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 importFeatureGroupsXCMS3 and importFeatureGroupsTable. 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: [importFeatureGroupsXCMS](#), [importFeatureGroupsXCMS3](#), [importFeatureGroupsKPIC2](#), [importFeatureGroupsTable](#), [importFeatureGroupsBruker](#), [importFeatureGroupsBrukerTASQ](#), [importFeatureGroupsEnviMass](#)
[groupFeatures](#) to group features.

```
importFeatureGroupsBrukerPA
```

Imports feature groups from Bruker ProfileAnalysis

Description

Imports a 'bucket table' produced by Bruker ProfileAnalysis (PA)

Usage

```
importFeatureGroupsBrukerPA(  
  input,  
  feat,  
  rtWindow = defaultLim("retention", "medium"),  
  mzWindow = defaultLim("mz", "medium"),  
  intWindow = 5,  
  warn = TRUE  
)
```

Arguments

input	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(  
  input,  
  analysisInfo,  
  clusterRTWindow = defaultLim("retention", "medium")  
)
```

Arguments

<code>input</code>	The file path to an Excel export of the Global results table from TASQ, converted to '.csv' format.
<code>analysisInfo</code>	A <code>data.frame</code> (or <code>data.table</code>) with Analysis information .
<code>clusterRTWindow</code>	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(input, feat, positive)
```

Arguments

input	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 feature groups from KPIC2

Usage

```
importFeatureGroupsKPIC2(input, analysisInfo)
```

Arguments

input	A grouped PIC set object (<i>e.g.</i> as returned by KPIC::PICset.group).
analysisInfo	A data.frame (or data.table) with Analysis information .

Details

This function imports data from KPIC2. This function is called when calling `importFeatureGroups` with `type="kpic2"`.

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

[importFeatureGroups](#) for more details and other algorithms.

[groupFeatures](#)

`importFeatureGroupsTable`*Import feature groups from a table*

Description

This function imports grouped features from a table, which can be either a data frame or a path to a file with tabular data (e.g. '.csv').

Usage

```
importFeatureGroupsTable(  
  input,  
  analysisInfo,  
  addCols = NULL,  
  groupAlgo,  
  groupArgs = NULL  
)
```

Arguments

<code>input</code>	The input to be imported: either a <code>data.frame</code> , <code>data.table</code> or file path to a file that can be imported with fread .
<code>analysisInfo</code>	A <code>data.frame</code> (or <code>data.table</code>) with Analysis information .
<code>addCols</code>	Passed to importFeaturesTable .
<code>groupAlgo, groupArgs</code>	(sets workflow) The grouping algorithm and a list with parameters that is used to re-group features when adduct assignments are changed, e.g. by selections . Hence, these are often unused and dummy values can be set here, such as <code>groupAlgo="openms"</code> , <code>groupArgs = list()</code> .

Details

This function imports data from a table. This function is called when calling `importFeatureGroups` with `type="table"`.

This function can be used to import feature group data from a table generated by the [as.data.table](#) (and `as.data.frame`) function, or the output from any other feature detection and grouping software. The column format mostly follows the format used by the `as.data.table` function with the `features` argument set to `TRUE`.

The function first imports the feature data from the table using [importFeaturesTable](#). Please see its documentation for an overview of the columns that are expected in the input table.

In addition to the columns for feature data, the following columns are expected in the input table:

- `group`: a string naming the feature group in which the feature is part of. Values in this column should be unique per analysis. (**mandatory**)

- `group_ret,group_mz`: the retention time (in seconds) and m/z assigned to the feature group. Will be calculated from mean values of feature data if missing. (**optional**)

For data from an [IMS workflow](#), the following additional columns are relevant:

- `group_mobility,group_ccs`: the mobility and CCS assigned to the feature group. Will be calculated from mean values of feature data if missing. (**optional**)
- `ims_precursor_group`: a string naming the IMS precursor feature group of the feature group (NA if none or not a IMS feature). (**optional**)

For data from a [sets workflow](#), the following additional columns are relevant:

- `group_ion_mz-<set>,group_neutralMass,group_adduct-<set>`: the ion m/z, neutral mass and adduct assigned to the feature group. Columns should be present for each set (*e.g.* `ion_mz-positive`). Missing columns will be calculated from feature data. **NOTE** If feature columns are missing, then the group columns will be used to create them.

Value

An object derived from the class `featureGroupsSet` (if the imported data is from a [sets workflow](#)) or `featureGroups` object otherwise.

Note

This function does not yet allow importing more advanced feature group properties, such as normalized intensities and predicted concentrations.

See Also

[importFeatureGroups](#) for more details and other algorithms.

[importFeaturesTable](#) for importing features from a table. [as.data.table](#) for converting feature groups data to a `data.table` format. [groupFeatures](#) to generate feature groups.

```
importFeatureGroupsXCMS
```

Imports feature groups from XCMS (old interface)

Description

Imports feature groups from XCMS (old interface)

Usage

```
importFeatureGroupsXCMS(input, analysisInfo)
```

Arguments

<code>input</code>	An <code>xcmsSet</code> object.
<code>analysisInfo</code>	A <code>data.frame</code> (or <code>data.table</code>) with Analysis information .

Details

This function imports data from XCMS. This function is called when calling importFeatureGroups with type="xcms".

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

[importFeatureGroups](#) for more details and other algorithms.
[groupFeaturesXCMS](#)

importFeatureGroupsXCMS3

Imports feature groups from XCMS (new interface)

Description

Imports feature groups from XCMS (new interface)

Usage

```
importFeatureGroupsXCMS3(input, analysisInfo)
```

Arguments

input	An XCMSnExp object.
analysisInfo	A data.frame (or data.table) with Analysis information .

Details

This function imports data from XCMS3. This function is called when calling importFeatureGroups with type="xcms3".

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

[importFeatureGroups](#) for more details and other algorithms.

[groupFeaturesXCMS3](#)

importFeatures	<i>Import features</i>
----------------	------------------------

Description

Generic function to import features produced by other software.

Usage

```
importFeatures(input, type, ...)
```

Arguments

input	The input object or path that should be imported. See the algorithm specific functions for more details.
type	What type of data should be imported: "xcms", "xcms3", "kpic2", "table", 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 importFeaturesTable. 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](#), [importFeaturesTable](#), [importFeaturesEnviMass](#) [findFeatures](#) to find new features.

`importFeaturesEnviMass`*Imports features from enviMass*

Description

Imports features from a project generated by the **enviMass** package.

Usage

```
importFeaturesEnviMass(input, analysisInfo)
```

Arguments

<code>input</code>	The path of the <code>enviMass</code> project.
<code>analysisInfo</code>	A <code>data.frame</code> (or <code>data.table</code>) with Analysis information .

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	<i>Imports features from KPIC2</i>
---------------------	------------------------------------

Description

Imports feature data generated by the **KPIC2** package.

Usage

```
importFeaturesKPIC2(input, analysisInfo)
```

Arguments

input	A list with a pics objects obtained with getPIC or getPIC.kmeans for each analysis.
analysisInfo	A data.frame (or data.table) 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.

importFeaturesTable	<i>Import features from a table</i>
---------------------	-------------------------------------

Description

This function imports features from a table, which can be either a data frame or a path to a file with tabular data (e.g. '.csv').

Usage

```
importFeaturesTable(input, analysisInfo, addCols = NULL)
```

Arguments

input	The input to be imported: either a data.frame, data.table or file path to a file that can be imported with fread .
analysisInfo	A data.frame (or data.table) with Analysis information .
addCols	A character vector with additional columns that should be included in the imported features object. These columns are not used by patRoan and should not interfere with internally used columns (these will be ignored with a warning if they do).

Details

This function imports data from a table. This function is called when calling importFeatures with type="table".

This function can be used to import features from a table generated by the [as.data.table](#) (and [as.data.frame](#)) function, or the output from any other feature detection software. The column format mostly follows the format used by the [as.data.table](#) function.

The following columns must be present:

- analysis: the analysis name (corresponding the the [analysis information](#)).
- ret: the retention time of the feature (in seconds).
- mz: the m/z value of the feature.
- intensity: the peak intensity of the feature.

The following columns are optional, but usually recommended:

- ID: the feature ID. Should be unique across the features from the same analysis. If not present, a sequential ID is automatically generated.
- area: the peak area of the feature. If not present, it is calculated as 2.5*intensity.
- retmin and retmax: the minimum and maximum retention time range of the feature. If not present, they are derived by subtraction or addition of ret by defaultLim("retention", "narrow") (see [limits](#)).

- `mzmin` and `mzmax`: the minimum and maximum m/z range of the feature. If not present, they are derived by subtraction or addition of `mz` by `defaultLim("mz", "narrow")` (see [limits](#)).

If a mobility column is present, it is assumed that the features are from an [IMS workflow](#). In this case

- `mobility`: specifies the mobility of the feature.
- `mobmin` and `mobmax`: the minimum and maximum mobility range of the feature. If not present, they are derived by subtraction or addition of mobility by `defaultLim("mobility", "narrow")` (see [limits](#)). (**optional**)
- `mob_area` and `mob_intensity`: the peak area and intensity of the peak in the mobilogram for the feature. (**optional**)
- `ims_precursor_ID`: the ID of the IMS precursor in a [post mobility assignment](#) workflow. (**optional**)
- `mob_assign_method`: a string that names the method used to assign the mobility to the feature. (**optional**)

If a set column is present, it is assumed that the features are from a [sets workflow](#). In this case

- `set`: specifies the set name in which the feature is present.
- `mz` and `ion_mz`: specify the *neutral mass* and *ionized m/z* of the feature, respectively.
- `adduct`: specifies the [adduct](#) of the feature (generic textual format), *e.g.* "[M+H]⁺".

Value

An object derived from the class [featuresSet](#) (if the imported features are from a [sets workflow](#)) or [features](#) object otherwise.

Note

The "set" column in the [analysis information](#) is not used when importing data (this column is present when obtaining the analysis information from a sets object with [analysisInfo](#)).

See Also

[importFeatures](#) for more details and other algorithms.

[importFeatureGroupsTable](#) to import feature group data from a table. [as.data.table](#) for converting features to a `data.table` format. [findFeatures](#) to generate feature data.

importFeaturesXCMS	<i>Imports features from XCMS (old interface)</i>
--------------------	---

Description

Imports feature data generated with the legacy [xcmsSet](#) function from the **xcms** package.

Usage

```
importFeaturesXCMS(input, analysisInfo)
```

Arguments

input	An xcmsSet object.
analysisInfo	A <code>data.frame</code> (or <code>data.table</code>) 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	<i>Imports features from XCMS (new interface)</i>
---------------------	---

Description

Imports feature data generated from an existing [XCMSnExp](#) object generated by the **xcms** package.

Usage

```
importFeaturesXCMS3(input, analysisInfo)
```

Arguments

input	An XCMSnExp object.
analysisInfo	A <code>data.frame</code> (or <code>data.table</code>) 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.

installC3SDB	<i>Automatically installs C3SDB</i>
--------------	-------------------------------------

Description

Automatically installs the **C3SDB** Python package

Usage

```
installC3SDB(envname = "patRoon-C3SDB", clearEnv = FALSE, ...)
```

Arguments

envname	The name of the virtual Python environment to install C3SDB into. Passed to reticulate::py_install . Set to NULL to not set any virtual Python environment.
clearEnv	Set to TRUE to remove the virtual environment if it already exists (using reticulate::virtualenv_remove). Ignored if envname is NULL.
...	Further arguments passed to reticulate::py_install .

Details

This function uses **reticulate** to install the **C3SDB** Python package in a virtual environment.

References

Ross DH, Cho JH, Xu L (2020). "Breaking Down Structural Diversity for Comprehensive Prediction of Ion-Neutral Collision Cross Sections." *Analytical Chemistry*, **92**(6), 4548–4557. ISSN 1520-6882, doi:10.1021/acs.analchem.9b05772, <http://dx.doi.org/10.1021/acs.analchem.9b05772>.

installTIMSCONVERT	<i>Automatically installs TIMSCONVERT</i>
--------------------	---

Description

Automatically installs the **TIMSCONVERT** Python package

Usage

```
installTIMSCONVERT(envname = "patRoon-TIMSCONVERT", clearEnv = FALSE, ...)
```

Arguments

envname	The name of the virtual Python environment to install TIMSCONVERT into. Passed to reticulate::py_install . Set to NULL to not set any virtual Python environment.
clearEnv	Set to TRUE to remove the virtual environment if it already exists (using reticulate::virtualenv_remove). Ignored if envname is NULL.
...	Further arguments passed to reticulate::py_install .

Details

This function uses **reticulate** to install the **TIMSCONVERT** Python package in a virtual environment.

References

Luu GT, Freitas MA, Lizama-Chamu I, McCaughey CS, Sanchez LM, Wang M (2022). “TIMSCONVERT: a workflow to convert trapped ion mobility data to open data formats.” *Bioinformatics*, **38**(16), 4046–4047. ISSN 1367-4811, doi:10.1093/bioinformatics/btac419, <http://dx.doi.org/10.1093/bioinformatics/btac419>.

kpic2-conv

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,
  IMS = FALSE,
  EICParams = getDefEICParams(window = 0)
)

## 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 (<i>e.g.</i> file paths) is used in the returned object.
IMS	(IMS workflow) Specifies which feature groups are considered for export in IMS workflows. The following options are valid: <ul style="list-style-type: none"> • "both": Selects IMS and non-IMS features. • "maybe": Selects non-IMS features and IMS features without assigned IMS precursor.

- FALSE: Selects only non-IMS features.
- TRUE: Selects only IMS features.

This should be kept FALSE as **KPIC2** export currently does not support IMS features.

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

Details

The conversion process will introduce some dummy values for metadata not present in **patRoan** objects. If the features object was generated with **KPIC2** and no post mobility assignment was performed, then no conversion is performed and the original **KPIC2** object will be returned.

Use of raw HRMS data

The [raw data interface](#) of **patRoan** is used by getPICSet to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

launchEICGUI

Launch EIC/EIM GUI

Description

Launches a Shiny app for generating and plotting EICs and EIMs.

Usage

```
## S4 method for signature 'data.frame'
launchEICGUI(obj, suspects = NULL, adduct = NULL)
```

```
## S4 method for signature 'features'
launchEICGUI(obj)
```

```
## S4 method for signature 'featureGroups'
launchEICGUI(obj)
```

Arguments

obj	For data.frame method: analysis information data.frame. For other methods: features/featureGroups object.
suspects	Optional suspect list data.frame (data.frame method only).
adduct	An adduct object (or something that can be converted to it with as.adduct). Examples: "[M-H]-", "[M+Na]+".

Details

These tools were originally developed as internal tools to debug and explore IMS related algorithms. They were mostly generated with LLMs, hence, may contain nonsense and bugs. Nevertheless, they may be useful for users to explore EICs and EIMs interactively. Please use with caution and report any issues.

Value

A Shiny app object.

limits	<i>Default limits and tolerances</i>
--------	--------------------------------------

Description

Get and configure default limits and tolerances used by **patRoön**.

Usage

```
defaultLim(category, level)

getLimIMS()

genLimitsFile(out = "limits.yml", IMS = "bruker")
```

Arguments

category, level	The category and level of the limit to be returned. See the detail sections below. For mobility related limits, the category="mobility" (instead of instrument specific categories) should be used.
out	The full output path for the new file.
IMS	The IMS instrument type. This sets the IMS variable in the general section. Valid values are "bruker" and "agilent".

Details

Tolerances for retention times, *m/z* values and other numerical limits and tolerances are widely used in **patRoön** to process data. Their defaults used to be hardcoded directly as defaults to function arguments. Since version 3.0 the defaults are centralized in a 'limits YAML' file. This simplifies their configuration and makes it easier to switch the defaults between different HRMS instruments.

The limits configuration file is taken from one of the following locations (in order):

- the path specified in the patRoön.path.limits option (if specified)
- the 'limits.yml' file in the current working directory (if present)
- the default 'limits.yml' file embedded in **patRoön**

defaultLim returns the limits for a specific category and tolerance level.

getLimIMS returns the type of IMS instrument specified in the limits configuration file. This is used to determine which mobility limits to use, and is also used in some other functions to determine the default behavior for IMS related processing.

genLimitsFile generates a new 'limits.yml' configuration file in the specified path. The file is created with the defaults embedded in **patRoorn** (see details below). Generating a custom limits is primarily useful for non-Bruker IMS workflows.

limits YAML file format

The limits are configured in a simple 'YAML' file. A brief summary of the format is given here.

The **general** section has the IMS field which specifies the type of instrument used. Currently this is either *bruker* or *agilent*.

The next sections describe absolute and relative (suffixed by *_rel*) limits and tolerances for retention, *m/z*, mobility and CCS data. These are divided into several tolerance levels: *very_narrow*, *narrow*, *medium* and *wide*. In general, the very narrow tolerances are used to compare data that should be equivalent, but may be slightly different due to *e.g.* small rounding errors. The narrow tolerances are generally sufficient when only small deviations are expected, *e.g.* comparing different *m/z* values from well calibrated HRMS instruments. The medium tolerances are generally used when a slightly larger tolerance is needed, *e.g.* when *m/z* values from raw (non-averaged) spectra. The wide values are mainly used for plot limits, *e.g.* to provide a reasonable zoom-out. The sections only define values for the tolerance levels actually used in **patRoorn**.

The *mobility_bruker* and *mobility_agilent* sections specify the default mobility limits for Bruker and Agilent systems, respectively. Which are used is set by the IMS variable in the **general** section.

To see which limits are used in which functions, please refer to the Usage section of the respective functions, specifically how the `defaultLim` function is used to assign function argument defaults.

NOTE: the choice between using the narrow and medium tolerance is not always clear, and there is some inconsistency in its use throughout **patRoorn** (primarily due to legacy code or keeping defaults from external algorithms).

Default limits YAML file

```
general:
  version: 1
  IMS: bruker
retention:
  very_narrow: 2
  narrow: 6
  medium: 12
  wide: 30
mz:
  very_narrow: 0.001
  narrow: 0.002
  medium: 0.005
  wide: 0.02
  narrow_rel: 5
```

```
    medium_rel: 10
mobility_bruker:
    very_narrow: 0.01
    narrow: 0.02
    medium: 0.04
    wide: 0.2
    medium_rel: 0.05
mobility_agilent:
    very_narrow: 0.1
    narrow: 0.2
    medium: 0.4
    wide: 2
    medium_rel: 0.05
CCS:
    medium: 10
    medium_rel: 0.05
```

Note

Most of the defaults were derived with Bruker TOF HRMS instrumentation in mind, but should be reasonable with minor adjustments for others.

loadMSLibrary	<i>Loading of MS library data</i>
---------------	-----------------------------------

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 = defaultLim("mz", "narrow"),  
  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 [generateCompoundLibrary](#).

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 **patRoan**.
- 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 **patRoan** (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 formulae 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 (2026). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.1, <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	<i>Load MS library data from MSP files</i>
------------------	--

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 = defaultLim("mz", "narrow"),
  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 **Repp** 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 **patRoan**.
- 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 **patRoön** (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 formulae 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 (2026). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.1, <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.

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 <i>set names</i> , 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 be 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.

MSConversion

MS data conversion

Description

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

Usage

```
getMSConversionTypes(algorithm, direction)

getMSConversionFormats(algorithm, direction, type = NULL)

convertMSFilesPWiz(
  inFiles,
  outFiles,
  formatTo = "mzML",
  centroid = TRUE,
  IMS = FALSE,
  minIntensity = 0,
  filters = NULL,
  extraOpts = NULL,
  PWizBatchSize = 1
```

```
)

convertMSFilesOpenMS(inFiles, outFiles, formatTo = "mzML", extraOpts = NULL)

convertMSFilesBruker(inFiles, outFiles, formatTo = "mzML", centroid = TRUE)

convertMSFilesIMSCollapse(
  inFiles,
  outFiles,
  typeFrom,
  formatTo = "mzML",
  mzRange = NULL,
  mobilityRange = NULL,
  smoothWindow = 0,
  halfWindow = 2,
  maxGap = 0.005,
  clusterMethod = "distance_mean",
  mzWindow = defaultLim("mz", "medium"),
  minIntensityIMS = 0,
  includeMSMS = FALSE,
  ...
)

convertMSFilesTIMSCONVERT(
  inFiles,
  outFiles,
  formatTo = "mzML",
  centroid = TRUE,
  centroidRaw = FALSE,
  IMS = FALSE,
  extraOpts = NULL,
  virtualenv = "patRoon-TIMSCONVERT"
)

convertMSFilesPaths(
  files,
  formatFrom,
  formatTo = "mzML",
  outPath = NULL,
  dirs = TRUE,
  overwrite = FALSE,
  algorithm = "pwiz",
  ...
)

convertMSFiles(
  anaInfo,
  typeFrom = "raw",
```

```

    typeTo = "centroid",
    formatFrom,
    formatTo = "mzML",
    overwrite = FALSE,
    algorithm = "pwiz",
    centroidVendor = TRUE,
    ...
)

```

Arguments

algorithm	Either "pwiz" (ProteoWizard), "openms", "bruker" (Bruker DataAnalysis), "imscollapse" or "timsconvert".
direction	A character specifying the direction of conversion. Either "input" or "output".
type, typeFrom, typeTo	The type of the input or output files. See <code>getMSConversionTypes</code> for the supported types.
inFiles, outFiles	A character vector with input and output files, respectively. Lengths and order should be the same.
centroid	Set to TRUE to perform centroiding. For <code>convertMSFilesPWiz</code> : the value may be "vendor" to perform centroiding with the vendor algorithm or "cwt" to use ProteoWizard's wavelet algorithm.
IMS	How to handle IMS data. For <code>convertMSFilesPWiz</code> : if TRUE then IMS data is exported and spectra for each IMS frame are combined into a single spectrum (using the <code>--combineIonMobilitySpectra</code> option), which is the format supported by patRoan . Set to NA to collapse the IMS data by scan summing, which mimics 'regular' HRMS data. Set to FALSE for non-IMS data. NOTE : do not set <code>IMS=FALSE</code> if the data has IMS data. This will result in very large files where MS spectra are not combined by frame, which cannot be properly read by patRoan . For <code>convertMSFilesTIMSCONVERT</code> : set to TRUE to keep IMS data or FALSE to exclude IMS data to mimic 'regular' LC-MS data.
minIntensity	The minimum intensity of the mass peaks to be kept. Applying an intensity threshold is especially beneficial to reduce export file size when there are a lot of zero or very low intensity mass peaks. NOTE this currently does <i>not</i> work well with IMS data.
filters	A character vector specifying one or more filters to <code>msconvert</code> . The elements of the specified vector are directly passed to the <code>--filter</code> option (see here)
extraOpts	A character vector specifying any extra command line parameters passed to <code>msconvert</code> or <code>FileConverter</code> . Set to NULL to ignore. For options: see FileConverter and msconvert .
PWizBatchSize	The number of analyses to process by a single call to <code>msconvert</code> . Usually a value of one is most efficient. Set to zero to run all analyses all at once from a single call.

mzRange, mobilityRange	A two sized vector specifying the m/z and mobility range to be exported, respectively. Set to NULL to export the full range.
smoothWindow, halfWindow, maxGap	Centroiding parameters: see getDefAvgPListParams for details. NOTE: As described there, maxGap may need to be increased for Agilent instruments (e.g. '0.01').
clusterMethod, mzWindow	The clustering method and window (see clustering parameters) used to find and combine MS/MS spectra of precursors with close m/z.
minIntensityIMS	The minimum intensity for MS peaks in raw data.
includeMSMS	Set to TRUE to include MS/MS spectra in the output. For IMS workflows where IMS data is only collapsed to produce compatible data files for feature detection, MS/MS data are not needed and can be excluded to reduce computational times and file sizes. Setting includeMSMS=TRUE is primarily intended to perform 'classical LC-MS workflows' with IMS data.
...	For convertMSFilesIMSCollapse: further arguments passed to mzR::writeMSData . For convertMSFilesPaths and convertMSFiles: further arguments passed to algorithm specific conversion functions.
centroidRaw	Only applicable if IMS=FALSE. Sets the mode parameter of TIMSCONVERT: raw if centroidRaw=TRUE or centroid if centroidRaw=FALSE. See https://gtluu.github.io/timsconvert/local.html#notes-on-mode-parameter for more details.
virtualenv	The virtual Python environment in which TIMSCONVERT is installed. This is passed to reticulate::use_virtualenv , which will ensure that the TIMSCONVERT command line utility can be found by patRoön . Set to NULL to skip this step.
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.
formatFrom, formatTo	The input or output format. See getMSConversionFormats for the supported formats.
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.
overwrite	Should existing destination file be overwritten (TRUE) or not (FALSE)?
anaInfo	An analysis info table that is used to retrieve the input files. The paths set by path_centroid, path_profile and path_ims are used to determine the output directories. This function automatically determines if and how centroiding and IMS conversions should be applied.
centroidVendor	Only for algorithm="pwiz": whether centroiding should be performed with vendor algorithms.

Details

`getMSConversionTypes` returns a character with all supported input or output conversion types for an algorithm.

`getMSConversionFormats` returns a character with all supported input or output conversion formats for an algorithm, optionally filtered by the given type.

`convertMSFilesPWiz` converts and pre-treats HRMS data with the `msconvert` tool from [ProteoWizard](#).

`convertMSFilesOpenMS` converts HRMS data with the `FileConvert` tool of [OpenMS](#).

`convertMSFilesBruker` converts and pre-treats Bruker HRMS data with Bruker DataAnalysis. Note that TIMS data currently is not supported.

`convertMSFilesIMSCollapse` is used to convert IMS data to data that mimics 'regular' HRMS data by collapsing the IMS dimension. The raw data interface of [patRoön](#) first sums up all spectra within each IMS frame, performs centroiding and finally exports the resulting data with the `mzR::writeMSData` function. Several thresholds can be set to speed up the conversion process and reduce noise, but care should be taken that no mass peaks of interest are lost.

`convertMSFilesTIMSCONVERT` converts and pre-treats TIMS data with [TIMSCONVERT](#). The `installTIMSCONVERT` function can be used to automatically install TIMSCONVERT.

`convertMSFilesPaths` is a wrapper function that simplifies the use of algorithm specific MS conversion functions, such as `convertMSFilesPWiz`, and `convertMSFilesTIMSCONVERT`.

`convertMSFiles` is a wrapper function that simplifies the use of `convertMSFilesPaths`.

Parallelization

`convertMSFilesPWiz`, `convertMSFilesOpenMS` and `convertMSFilesTIMSCONVERT` uses multi-processing to parallelize computations. Please see the parallelization section in the handbook for more details and [patRoön options](#) for configuration options.

Use of raw HRMS data

The [raw data interface](#) of [patRoön](#) is used by `convertMSFilesIMSCollapse` to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

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.

Luu GT, Freitas MA, Lizama-Chamu I, McCaughey CS, Sanchez LM, Wang M (2022). “TIM-SCONVERT: a workflow to convert trapped ion mobility data to open data formats.” *Bioinformatics*, **38**(16), 4046–4047. ISSN 1367-4811, doi:10.1093/bioinformatics/btac419, <http://dx.doi.org/10.1093/bioinformatics/btac419>.

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.

msdata

Interface for HRMS and IMS-HRMS raw data

Description

Description, configuration and utilities for the raw (IMS-)HRMS data interface of **patRoön**.

Usage

```
availableBackends(
  anaInfo = NULL,
  needTypes = NULL,
  checkOption = TRUE,
  verbose = TRUE
)
```

Arguments

anaInfo	Optional. If not NULL then anaInfo should be a analysis information table, and only those backends that can read each of the analyses are returned.
needTypes	Only applicable if anaInfo is set: should be "centroid", "profile" and/or "ims" to filter file types.
checkOption	Set to TRUE to only consider backends that are included in the patRoan.MS.backends option.
verbose	Set to TRUE to print the status of each backend.

Details

Version 3.0 of **patRoan** introduced an extensible and highly optimized interface to read raw data from HRMS and IMS-HRMS instruments. This interface supports chooseable 'backends' which perform the reading of file data from various formats. Subsequent steps such as the formation of extracted ion chromatograms, mobilograms and collection and averaging of mass spectra are then performed in patRoan. The interface is largely coded in C++ (using **Rcpp**), uses **OpenMP** parallelization and applies several other optimization strategies to make it suitable to rapidly process large amounts of raw data, *e.g.* as encountered in IMS-HRMS workflow.

The following backends for reading (IMS-)HRMS data are currently available:

- "opentims": uses the **OpenTIMS** library to read Bruker TIMS data. This backend supports very fast reading of raw instrument '.d' data files directly, and therefore does not require any file conversions. This backend only supports 64 bit Windows and Linux systems. See the Backend installation section below installation instructions.
- "mzr": uses the **mzR** package to read '.mzML' and '.mzXML' files. This package was more or less the default in **patRoan** prior to 3.0, and due to its popularity and age is a stable and well tested option. The mzr backend currently reads the complete analysis file at once, which makes it more RAM intensive compared to other backends. The read data is cached to speed up any subsequent operations that require the file data. This backend currently does not support IMS data. Since **mzR** is a dependency of **patRoan**, no additional installation is necessary.
- "mstoolkit": Uses the **MSToolKit** C++ library to read '.mzML' and '.mzXML' files, including IMS-HRMS data. The MSToolKit library has been developed for many years, and was recently updated with IMS-HRMS support. See the Backend installation section below installation instructions.
- "streamcraft": Uses the **StreamCraft** C++ library to read '.mzML' and '.mzXML' files, including IMS-HRMS data. The StreamCraft library is still young and somewhat experimental. The library is integrated within **patRoan** and therefore does not require any further installation.

The `availableBackends` function is used to query the available backends on the system.

Value

`availableBackends` returns (invisibly) a character vector with the names of the available backends.

Configuration

The following package options influence the behavior of raw data interface:

- `patRoos.MS.backends`: A character vector with the names of the backends that may be chosen. The default is all backends. The first backend will be chosen that is available, is able to read at least one of the available analysis file types and formats (as configured by the [analysis information](#)) and supports IMS if needed.
- `patRoos.MS.preferIMS`: A logical value that indicates whether the IMS data should be preferred, even if the processing step does not require IMS data and non-IMS data is also available. Setting this to TRUE probably result in some additional computational overhead, but may avoid any inconsistencies between the IMS data and non-IMS data that may have been introduced during the conversion step of the latter. This option is only relevant for the `mstoolkit` and `streamcraft` backends (and if one of these backends is actually used).
- `patRoos.threads`: An integer value that indicates the number of threads to use for parallelization (multithreading). The default is determined from the number of physical cores of the system (obtained with the [parallel::detectCores](#) function).
- `patRoos.path.TDFSdk`: The file path to the Bruker TDF-SDK library file. See the Backend installation section below.

Backend installation

The `opentims` backend requires the `'win64/timsdata.dll'` (Windows) or `'linux64/libtimsdata.so'` (Linux) file from the TDF-SDK from **Bruker** (requires login). The **patRoosExt** package makes these files automatically available for **patRoos**. Otherwise the `patRoos.path.TDFSdk` option should be manually set to the file path of the `'timsdata.dll'` or `'linux64/libtimsdata.so'` file.

When **patRoos** is installed from source, e.g. on Linux/macOS systems or when using `remotes::install_github` for installation, then the <https://github.com/rickhelmus/Rmstoolkitlib> R package must be installed in advance.

The `availableBackends` function can be used to verify if the dependencies for these backends are met.

References

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 (2026). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.1, <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.

Dagum L, Menon R (1998). “OpenMP: an industry standard API for shared-memory programming.” *IEEE Computational Science and Engineering*, **5**(1), 46–55. doi:10.1109/99.660313.

Łącki MK, Startek MP, Brehmer S, Distler U, Tenzer S (2021). “OpenTIMS, TimsPy, and TimsR: Open and Easy Access to timsTOF Raw Data.” *Journal of Proteome Research*, **20**(4), 2122–2129. ISSN 1535-3907, doi:10.1021/acs.jproteome.0c00962, <http://dx.doi.org/10.1021/acs.jproteome.0c00962>.

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.

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

<code>x, obj, object</code>	MSLibrary object to be accessed.
<code>i</code>	For <code>[[</code> : A numeric or character value which is used to select records by their index or name, respectively (for the order/names see <code>names()</code>). For <code>:</code> : Can also be logical to perform logical selection (similar to regular vectors). If missing all records are selected. For <code>[[</code> : should be a scalar value.
<code>...</code>	Unused.
<code>drop, j</code>	ignored.
<code>name</code>	The record name (partially matched).
<code>properties</code>	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 <i>e.g.</i> <code>names(records)</code> . Example: <code>properties=list(Instrument_type=c("LC-ESI-QTOF", "LC-ESI-TOF"))</code> . Set to <code>NULL</code> to ignore.
<code>massRange</code>	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 <code>NULL</code> to ignore.
<code>mzRangeSpec</code>	Similar to the <code>massRange</code> argument, but removes any peaks from recorded mass spectra outside the given <i>m/z</i> 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 <i>no</i> 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, <i>e.g.</i> "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 <code>Suspect</code> 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 <code>Suspect</code> 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 <code>convertToSuspects</code> . For efficiency reasons it is recommended to set this to TRUE. See the <code>Validating and calculating chemical properties</code> 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 (<i>e.g.</i> [M+H] ⁺ , [M-H] ⁻). See the <code>Validating and calculating chemical properties</code> 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 formulae 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 (2026). *Rcpp: Seamless R and C++ Integration*. R package version 1.1.1, <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](#)

MSPeakLists-class	<i>Class containing MS Peak Lists</i>
-------------------	---------------------------------------

Description

Contains all MS (and MS/MS where available) peak lists for a [featureGroups](#) object.

Usage

```
peakLists(obj, ...)  
  
averagedPeakLists(obj, ...)  
  
spectrumSimilarity(obj, ...)  
  
spectrumSimilarityIMS(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,
  MSLevel = 1:2,
  absMinIntensity = NULL,
  relMinIntensity = NULL,
  topMostPeaks = NULL,
  minPeaks = NULL,
  maxMZOverPrec = NULL,
  absMinAbundanceFeat = NULL,
  relMinAbundanceFeat = NULL,
  absMinAbundanceFGroup = NULL,
  relMinAbundanceFGroup = NULL,
  relMinCumIntensity = NULL,
  isolatePrec = NULL,
  deIsotope = FALSE,
  removeMZs = NULL,
  withMSMS = FALSE,
  annotatedBy = NULL,
  retainPrecursor = TRUE,
  mzWindow = defaultLim("mz", "medium"),
  reAverage = FALSE,
  negate = FALSE
)

## S4 method for signature 'MSPeakLists'
plotSpectrum(
  obj,

```

```

    groupName,
    analysis = NULL,
    MSLevel = 1,
    title = NULL,
    normalized = "multiple",
    specSimParams = getDefSpecSimParams(),
    xlim = NULL,
    ylim = NULL,
    showLegend = TRUE,
    ...
)

## 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 'MSPeakLists'
spectrumSimilarityIMS(obj, fGroups, doFGroups = TRUE, warn = TRUE, ...)

## S4 method for signature 'MSPeakListsSet'
analysisInfo(obj, df = FALSE)

## 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,
  ...,
  removeMZs = NULL,

```

```

    withMSMS = FALSE,
    annotatedBy = NULL,
    retainPrecursor = TRUE,
    mzWindow = defaultLim("mz", "medium"),
    reAverage = FALSE,
    negate = FALSE,
    sets = NULL
)

## S4 method for signature 'MSPeakListsSet'
plotSpectrum(
  obj,
  groupName,
  analysis = NULL,
  MSLevel = 1,
  title = NULL,
  normalized = "multiple",
  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

<code>obj, x, object</code>	The <code>MSPeakLists</code> object to access.
<code>...</code>	For the "[" operator: ignored. For delete: passed to the function specified as <code>j</code> .

	For plotSpectrum: passed to plot .
	For spectrumSimilarityIMS: passed to spectrumSimilarity
	For sets workflow methods: further arguments passed to the base MSPeakLists method.
i, j	For <code>[/[]</code> : A numeric or character value which is used to select analyses/feature groups by their index or name, respectively (for the order/names see <code>analyses()/groupNames()</code>). For <code>[]</code> : Can also be logical to perform logical selection (similar to regular vectors). If missing all analyses/feature groups are selected. For <code>[]</code> : 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 <code>data.table</code>), feature group name, analysis (NULL if averaged function specifies the peak list indices (rows) to be removed (specified as an integer or logical
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 <i>e.g.</i> 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 <i>m/z</i> 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 <code>k!=NULL</code> then deletions will <i>not</i> occur on group averaged peak lists. Otherwise, if <code>k=NULL</code> then deletion occurs on <i>both</i> group averaged and analysis specific peak lists.
MSLevel	The MS level for which data is plotted or filtered: '1' for regular MS, '2' for MSMS. For filter: can also be 1:2 to specify both.
absMinIntensity, relMinIntensity	Absolute/relative intensity threshold for peaks. Set to NULL for none.
topMostPeaks	Only consider this number of most intense peaks. Set to NULL to consider all.
minPeaks	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.
maxMZOverPrec	Any mass peaks with an <i>m/z</i> higher than this value (relative to the precursor) will be removed. Set to NULL to ignore.

absMinAbundanceFeat, relMinAbundanceFeat

The minimum absolute/relative abundance for a mass peak across spectra that are averaged for a feature. Setting reAverage determines if feature group peak lists are also filtered:

- reAverage=FALSE then this filter is also applied to feature group data, using the the mean averaged peak abundance from the peak lists in the group.
- reAverage=TRUE then this filter is *not* applied to the (regenerated) feature group data.

In most cases reAverage=TRUE makes more sense to avoid inconsistent filtering approaches between feature and feature group data.

Set to NULL to ignore.

absMinAbundanceFGroup, relMinAbundanceFGroup

The minimum absolute/relative abundance of a mass peak across spectra that are averaged for a feature group. Set to NULL to ignore.

relMinCumIntensity

The minimum relative cumulative intensity of the peaks. For instance, a value of '0.95' means that only the most intense peaks that together account for 95% of the total intensity are retained. 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).

deIsotope

Remove any isotopic peaks in 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 generateMSPeakLists does not (yet) support flagging of isotopes.

removeMZs

A set of m/z values to be removed from the peak lists. This is typically used to remove background peaks. The m/z values should be specified by either be a numeric vector or a data.frame with an mz column. The latter is returned by the [getBGMSMSPeaks](#) function, which attempts to automatically detect background peaks.

(**sets workflow**) Should be a list that specifies the m/z values to be removed (in above mentioned format) for each set. The order should match that of the sets in the MSPeakLists object.

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.

NOTE: the annotatedBy filter currently only supports filtering peak of feature groups (and not of features). Hence, this filter cannot be combined with reAverage=TRUE. Furthermore, if peak lists are re-averaged after application of this filter, any filtered results will be *undone*.

retainPrecursor

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.

mzWindow	The m/z window used to find peaks to be removed from the removeMZs filter.
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.
title	The title of the plot. If NULL a title will be automatically made.
normalized	Controls intensity normalization. Should be FALSE (don't normalize), TRUE (normalize) or "multiple" (only normalizes if multiple spectra are plotted).
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.
showLegend	Set to TRUE to show a legend.
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>i.e.</i> 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>i.e.</i> when no similarity could be calculated) to zero values.
doFGroups	Set to TRUE to compare spectra of feature groups, FALSE to compare spectra of features.
warn	Set to TRUE to show a warning when no relevant feature group data is found.
df	If TRUE then a data.frame is returned, otherwise a data.table is returned.
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.

`spectrumSimilarityIMS` returns a `data.table` with spectral similarities for each IMS precursor and feature pair.

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.
- `spectrumSimilarityIMS(MSPeakLists)`: Calculates the spectral similarity between spectra from IMS features (or feature groups) and their IMS precursors in post mobility workflows (see [assignMobilities](#)).

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 propagated 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 considered. 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 `getDefaultIsolatePrecParams` 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 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 (sets argument).
- 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.* intensity filters (absMinIntensity/relMinIntensity), topMostPeaks and minPeaks. Furthermore, when the annotatedBy filter is applied, each set specific MS peak list is filtered by the annotation results from only that set. Finally, the removeMZs filter should be set for each set separately.
- 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.

newProject

*Easily create new **patRo**on 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.
----------	---

optimizationResult-class

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>i.e.</i> 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 <code>contour</code> (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)
```

pred-aggr-params	<i>Parameters to aggregate concentrations/toxicity values assigned to feature groups</i>
------------------	--

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, <i>e.g.</i> 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

```
calculateConcs(fGroups, ...)

## 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,
  pHAg,
  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,
  pHAg,
  concUnit = "ugL",
  calibConcUnit = concUnit
)

getQuantCalibFromScreening(
  fGroups,
  concs,
  areas = FALSE,
  average = FALSE,
  IMS = "maybe"
)
```

Arguments

fGroups	For predictRespFactors methods for feature annotations: The featureGroups object for which the annotations were performed.
---------	--

	For calculateConcs: The <code>featureGroups</code> object for which concentrations should be calculated.
	For getQuantCalibFromScreening: A feature groups object screened for the calibrants with <code>screenSuspects</code> .
...	(sets workflow) Further arguments passed to the non-sets workflow method.
featureAnn	A <code>featureAnnotations</code> object (e.g. <code>formulasSIRIUS</code> or <code>compounds</code>) which contains response factors. Optional if calculateConcs is called on suspect screening results (i.e. <code>featureGroupsScreening</code> 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 (<code>featureGroupsScreening</code>) or compound annotations (<code>compounds</code>).
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 future package. Please see the parallelization section in the handbook for more details.
type	Which types of predictions should be performed: should be "FP" (SIRIUS+CSI:FingerID fingerprints), "SMILES" or "both". Only relevant for <code>compoundsSIRIUS</code> method.
concs	A data.frame with concentration data. See the Calibration section below.
average	Set to TRUE to average intensity values within replicates.
IMS	(IMS workflow) Specifies which feature groups are considered for generating calibrant data in IMS workflows. The following options are valid: <ul style="list-style-type: none"> • "both": Selects IMS and non-IMS features. • "maybe": Selects non-IMS features and IMS features without assigned IMS precursor. • FALSE: Selects only non-IMS features.

- TRUE: Selects only IMS features.

Best to keep this "maybe", as calibration typically doesn't support IMS filtered data.

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 `featureGroups` 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 replicates (as defined in the [analysis information](#)). The concentration columns should be named after the corresponding replicate. Only those replicates that should be used for calibration need to be included. Furthermore, NA values can be used if a replicate 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.

In IMS workflows with post mobility assignments (see [assignMobilities](#)), the intensities of *IMS precursors* are used for the calculation of concentrations for IMS features (as calibration typically does not consider differences due to mobility filtering). However, the response factor assigned to IMS features are still used. This may yield small differences compared to workflows where concentrations are assigned prior to mobility assignments, as `assignMobilities` simply copies concentrations from IMS precursors to IMS features.

Note

The **redk** 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, Krueve 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

```
calculateTox(fGroups, ...)

## 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, LC50Mode = "static", concUnit = "ugL")

## 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.
...	(sets workflow) Further arguments passed to the non-sets workflow method.
featureAnn	A featureAnnotations object (<i>e.g.</i> formulasSIRIUS or compounds) which contains toxicities. Optional if calculateTox is called on suspect screening results (<i>i.e.</i> featureGroupsScreening method).
obj	The workflow object for which predictions should be performed, <i>e.g.</i> 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 future package. Please see the parallelization section in the handbook for more details.
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 **rdck** 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, Krue 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	<i>Prints all the package options of patRoan and their currently set values.</i>
------------------	--

Description

Prints all the package options of `patRoan` and their currently set values.

Usage

```
printPackageOpts()
```

reporting	<i>Report workflow data</i>
-----------	-----------------------------

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 = "patRoan"),  
  path = NULL,  
  EICParams = getDefEICParams(topMost = 1, topMostByReplicate = TRUE),  
  EIMParams = getDefEIMParams(topMost = 1, topMostByReplicate = TRUE),  
  specSimParams = getDefSpecSimParams(),  
  clearPath = FALSE,  
  openReport = TRUE,  
  parallel = FALSE,  
  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 = "patRoan"),  
  path = NULL,  
  EICParams = getDefEICParams(topMost = 1, topMostByReplicate = TRUE),  
  EIMParams = getDefEIMParams(topMost = 1, topMostByReplicate = TRUE),  
  specSimParams = getDefSpecSimParams(),
```



```

    clearPath = FALSE,
    openReport = TRUE,
    parallel = FALSE,
    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.
EIMParams	A named list with parameters used for extracted ion mobilogram (EIM) creation. See the EIM 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 future package. Please see the parallelization section in the handbook for more details. NOTE: parallelization is disabled by default, as it may slow down reporting on some systems (<i>e.g.</i> Windows) and under some circumstances. It is best to experiment with this setting to see if it speeds up report generation for your system and data.
overrideSettings	A list with settings that override those from the report settings file. Example: <code>overrideSettings=list(compounds=list(topMost=25))</code> .
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 various plots (*e.g.* chromatograms, (annotated) mass spectra and many more). This function uses functionality from other R packages, such as **rmarkdown**, **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
 - `version`: version of the settings file.
 - `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 generally smaller and easily shared, since only the ‘report.html’ needs to be copied. However, they are slower to generate and plots cannot be cached.
 - `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`.
 - mobilograms
 - * `large`, `small`, `features`: inclusion of mobilogram plots, see chromatograms above.
 - `intensityPlots`: inclusion of intensity trend plots.

- aggregateConcs, aggregateTox: function name used for concentration and toxicity aggregation, *e.g.* mean.
- 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 and which score fields to exclude from normalization; these are forwarded 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.
 - onlyUsedScorings: if true only scorings used by the current dataset are considered when normalizing or reporting compound scores.
- 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.

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by report to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

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`, `topMostByReplicate` 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 1.51, <https://yihui.org/knitr/>.

reporting-legacy

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(window = 20, topMost = 1, topMostByReplicate = TRUE),
    clearPath = 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
)

```

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 <i>not</i> be normalized. Set to NULL to normalize all scorings. Note that whether any normalization occurs is set by the <code>compoundsExclNormScores</code> , <code>formulasExclNormScores</code> 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 <i>mandatory</i> 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.

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.

Use of raw HRMS data

The [raw data interface](#) of **patRoön** is used by the report functions to process HRMS (or IMS-HRMS) data. Please see [its documentation](#) for more information on the supported formats and available configuration options.

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`, `topMostByReplicate` and `onlyPresent` [EIC parameters](#) may be ignored, *e.g.*, when generating overview plots.

See Also

`reporting`

retDir	<i>Retention order direction</i>
--------	----------------------------------

Description

Calculation of the relative retention order between a parent and its transformation product (TP).

Details

The relative retention order between a parent and its TP (retDir) is used throughout TP screening workflows for characterization and prioritization purposes. These are numeric values that hint what 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.

For TP data obtained with [generateTPs](#), the missing retDir values are automatically calculated based on the log P difference between the parent and TP. Here, a typical reversed phase separation is assumed, *i.e.* compounds with (significantly) lower log P values likely elute earlier. The minLogPDiff parameter of the [TPStructParams](#) argument sets the minimum log P difference to be considered significant.

For TP feature candidates that were linked by [generateComponentsTPs](#), the retDir values are calculated based on the retention time difference between the parent and TP feature groups. The minRTDiff argument sets the minimum difference to be considered significant.

References

Helmus R, Bagdonaitė I, de Voigt P, van Bommel MR, Schymanski EL, van Wezel AP, ter Laak TL (2025). “Comprehensive Mass Spectrometry Workflows to Systematically Elucidate Transformation Processes of Organic Micropollutants: A Case Study on the Photodegradation of Four Pharmaceuticals.” *Environmental Science & Technology*, **59**(7), 3723–3736. ISSN 1520-5851, doi:10.1021/acs.est.4c09121, <http://dx.doi.org/10.1021/acs.est.4c09121>.

sets-workflow	<i>Sets workflows</i>
---------------	-----------------------

Description

With sets workflows in **patRoön** 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 **patRoön** handbook.

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. Defaults to `defaultLim("mz", "medium")` (see [limits](#)).
- **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.

suspect-screening	<i>Target and suspect screening</i>
-------------------	-------------------------------------

Description

Utilities to screen for analytes with known or suspected identity.

Usage

```
screenSuspects(  
  fGroups,  
  suspects,  
  rtWindow = defaultLim("retention", "medium"),  
  mzWindow = defaultLim("mz", "medium"),  
  IMSMatchParams = NULL,  
  adduct = NULL,  
  skipInvalid = TRUE,  
  prefCalcChemProps = TRUE,  
  neutralChemProps = FALSE,  
  onlyHits = FALSE,  
  ...  
)  
  
## S4 method for signature 'featureGroups'  
screenSuspects(  
  fGroups,  
  suspects,  
  rtWindow,  
  mzWindow,  
  IMSMatchParams,  
  adduct,  
  skipInvalid,  
  prefCalcChemProps,  
  neutralChemProps,  
  onlyHits  
)  
  
## S4 method for signature 'featureGroupsScreening'  
screenSuspects(  
  fGroups,  
  suspects,  
  rtWindow,  
  mzWindow,  
  IMSMatchParams,  
  adduct,  
  skipInvalid,  
  prefCalcChemProps,
```

```

        neutralChemProps,
        onlyHits,
        amend = FALSE
    )

## S4 method for signature 'featureGroupsSet'
screenSuspects(
    fGroups,
    suspects,
    rtWindow,
    mzWindow,
    IMSMatchParams,
    adduct,
    skipInvalid,
    prefCalcChemProps,
    neutralChemProps,
    onlyHits
)

## S4 method for signature 'featureGroupsScreeningSet'
screenSuspects(
    fGroups,
    suspects,
    rtWindow,
    mzWindow,
    IMSMatchParams,
    adduct,
    skipInvalid,
    prefCalcChemProps,
    neutralChemProps,
    onlyHits,
    amend = FALSE
)

```

Arguments

fGroups	The featureGroups object that should be screened.
suspects	A <code>data.frame</code> with suspect information. See the <code>Suspect list</code> 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 (<i>e.g.</i> <code>suspects=list(positive=suspsPos, negative=suspsNeg)</code>).
rtWindow, mzWindow	The retention time window (in seconds) and <i>m/z</i> window that will be used for matching a suspect (+/- feature data).
IMSMatchParams	(IMS workflow) A list with parameters to be used for matching IMS data. See getIMSMatchParams for details and how to make such a parameter list.

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 <i>amended</i> to the original object.

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.

Value

`screenSuspects` returns a [featureGroupsScreening](#) object, which is a copy of the input `fGroups` object amended with additional screening information.

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 `estimateIDConfidence` to report detected MS/MS fragments and calculate identification levels. (**optional**)
- `mobility,CCS` The mobility or CCS value of the suspect. These values may be used to filter out suspects, see the `IMSMatchParams` argument. Multiple values for a single suspect can be specified by separating them with a semicolon(;). Adduct specific columns may be added by suffixing the adduct to the column name, *e.g.* `mobility_[M+H]+` and `CCS_[M-H]-`. (**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.

IMS reference assignment

If both adduct specific and non-adduct specific reference values are available, then non-adduct specific data is chosen (unless NA) as reference for the suspect hit. Otherwise, data is taken from the adduct specific data corresponding to the adduct assigned to the feature group (or adduct argument). If multiple mobility or CCS values for a suspect are specified in the suspect list, then the reference value is chosen which is the closest to that of the feature.

Validating and calculating chemical properties

Chemical properties such as SMILES, INCHIKEY and formulae 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.

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.

Note

`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`

TPLogicTransformations

Obtain default rules for metabolic logic

Description

This function returns a `data.frame` with the default rules for metabolic logic, which can be used by [generateTPsLogic](#) and [genFormulaTPLibrary](#).

Usage

```
TPLogicTransformations()
```

Value

A data.frame with columns describing each transformation rule.

Source

The table is 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.

transformationProducts-class

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

...
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](#).

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
 - transformationProductsAnnComp
 - transformationProductsBT
 - transformationProductsLibrary
 - * transformationProductsFormula
 - transformationProductsAnnForm
 - transformationProductsLibraryFormula
 - * transformationProductsLogic

See Also

The derived [transformationProductsStructure](#) class for more methods, [generateTPs](#) and [retDir](#).

transformationProductsAnnComp-class

Transformation products obtained from compound annotations

Description

Class to store results transformation products (TPs) obtained from compound annotations.

Usage

```
## S4 method for signature 'transformationProductsAnnComp'
filter(
  obj,
  ...,
  minFitFormula = 0,
  minFitCompound = 0,
  minSimSusp = 0,
  minFitCompOrSimSusp = c(0, 0),
  minTPScore = 0,
  topMost = NULL,
  verbose = TRUE,
  negate = FALSE
)
```

Arguments

obj	The transformationProductsAnnComp object that should be filtered.
...	Further arguments passed to the parent filter method .
minFitFormula, minFitCompound, minSimSusp, minFitCompOrSimSusp, minTPScore	Thresholds related to TP scoring. See generateTPsAnnComp for more details.
topMost	Only keep this number of top-most TPs (based on TPScore) for each parent/feature group combination. Set to NULL to skip this step.
verbose	If set to FALSE then no text output is shown.
negate	If TRUE then filters are performed in opposite manner.

Details

This class is derived from the [transformationProductsStructure](#) base class, please see its documentation for more details. Objects from this class are returned by [generateTPsAnnComp](#).

Methods (by generic)

- `filter(transformationProductsAnnComp)`: Performs rule-based filtering. Useful to simplify and clean-up the data.

S4 class hierarchy

- [transformationProductsStructure](#)
 - [transformationProductsAnnComp](#)

See Also

The base class [transformationProductsStructure](#) for more relevant methods and [generateTPsAnnComp](#)

transformationProductsAnnForm-class

Transformation products obtained from formula annotations

Description

Class to store results transformation products (TPs) obtained from formula annotations.

Usage

```
## S4 method for signature 'transformationProductsAnnForm'
filter(
  obj,
  ...,
  minFitFormula = 0,
  minTPScore = 0,
  topMost = NULL,
  verbose = TRUE,
  negate = FALSE
)
```

Arguments

<code>obj</code>	The transformationProductsAnnForm object that should be filtered.
<code>...</code>	Further arguments passed to the parent filter method .
<code>minFitFormula, minTPScore</code>	Thresholds related to TP scoring. See generateTPsAnnForm for more details.
<code>topMost</code>	Only keep this number of top-most TPs (based on TPScore) for each parent/feature group combination. Set to NULL to skip this step.
<code>verbose</code>	If set to FALSE then no text output is shown.
<code>negate</code>	If TRUE then filters are performed in opposite manner.

Details

This class is derived from the [transformationProductsFormula](#) base class, please see its documentation for more details. Objects from this class are returned by [generateTPsAnnForm](#).

Methods (by generic)

- `filter(transformationProductsAnnForm)`: Performs rule-based filtering. Useful to simplify and clean-up the data.

S4 class hierarchy

- [transformationProductsFormula](#)
 - [transformationProductsAnnForm](#)

See Also

The base class [transformationProductsFormula](#) for more relevant methods and [generateTPsAnnForm](#)

`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

<code>obj</code>	transformationProductsFormula derived object to be accessed
<code>which</code>	Either a character or integer vector with one or more names/indices of the parents to plot.
<code>components</code>	If specified (<i>i.e.</i> 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 <code>prune</code> and <code>onlyCompletePaths</code> arguments.
<code>prune</code>	If TRUE and <code>components</code> is set, then pathways without <i>any</i> detected TPs are not shown (pruned). See also the <code>onlyCompletePaths</code> and <code>components</code> arguments.
<code>onlyCompletePaths</code>	If TRUE and <code>components</code> is set, then only pathways are shown for which <i>all</i> TPs were detected. See also the <code>prune</code> and <code>components</code> arguments.
<code>width, height</code>	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](#)
 - * [transformationProductsAnnForm](#)
 - * [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 = NULL,
  nintersects = NA,
  upsetArgs = NULL
)

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

```

Arguments

<code>out</code>	The file name of the the output MetFrag database.
<code>includeParents</code>	Set to TRUE to include the parents in the database.
<code>obj, TPs</code>	<code>transformationProductsStructure</code> derived object to be accessed
<code>...</code>	For filter: Further argument passed to the base filter method . For <code>plotVenn</code> , <code>plotUpSet</code> and <code>consensus</code> : further (unique) <code>transformationProductsStructure</code> objects.
<code>removeParentIsomers</code>	If TRUE then TPs with an equal formula as their parent (isomers) are removed.
<code>removeTPIsomers</code>	If TRUE then all TPs with equal formula as any sibling TPs (isomers) are removed. Unlike <code>removeDuplicates</code> , <i>all</i> TP candidates are removed (including

the first match). This filter automatically sets `removeDuplicates=TRUE` so that TPs are only removed if with different structure.

<code>removeDuplicates</code>	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.
<code>minSimilarity</code>	Minimum structure similarity ('0-1') that a TP should have relative to its parent. This data is only available if the <code>calcSims</code> argument to <code>generateTPs</code> 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 <code>generateComponentsTPs</code>). Any values that are NA are removed (which only occur when a consensus was made from objects that not all have similarity information).
<code>verbose</code>	If set to FALSE then no text output is shown.
<code>negate</code>	If TRUE then filters are performed in opposite manner.
<code>which</code>	Either a character or integer vector with one or more names/indices of the parents to plot.
<code>components</code>	If specified (<i>i.e.</i> not NULL), a <code>componentsTPs</code> object that is used for matching the graph with screening results. The TPs that were found will be marked. See also the <code>prune</code> and <code>onlyCompletePaths</code> arguments.
<code>structuresMax</code>	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.
<code>prune</code>	If TRUE and <code>components</code> is set, then pathways without <i>any</i> detected TPs are not shown (pruned). See also the <code>onlyCompletePaths</code> and <code>components</code> arguments.
<code>onlyCompletePaths</code>	If TRUE and <code>components</code> is set, then only pathways are shown for which <i>all</i> TPs were detected. See also the <code>prune</code> and <code>components</code> arguments.
<code>width, height</code>	Passed to <code>visNetwork</code> .
<code>commonParents</code>	Only consider TPs from parents that are common to all compared objects.
<code>labels</code>	A character with names to use for labelling. If NULL labels are automatically generated.
<code>vennArgs</code>	A list with further arguments passed to VennDiagram plotting functions. Set to NULL to ignore.
<code>nsets, nintersects</code>	See <code>upset</code> . If <code>nsets == NULL</code> then it will be set to the number of compared items.
<code>upsetArgs</code>	A list with any further arguments to be passed to <code>upset</code> . Set to NULL to ignore.
<code>absMinAbundance, relMinAbundance</code>	Minimum absolute or relative ('0-1') abundance across objects for a result to be kept. For instance, <code>relMinAbundance=0.5</code> 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 <code>uniqueFrom</code> 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 <code>algorithm(obj)</code>). 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`
 - * `transformationProductsAnnComp`
 - * `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, Strobel 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	<i>Verifies if all dependencies are installed properly and instructs the user if this is not the case.</i>
--------------------	--

Description

Verifies if all dependencies are installed properly and instructs the user if this is not the case.

Usage

```
verifyDependencies()
```

withOpt	<i>Temporarily changes package options</i>
---------	--

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 `patRoan`.

Usage

```
withOpt(code, ..., prefix = "patRoan.")
```

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
 - transformationProductsAnnComp
 - transformationProductsBT
 - transformationProductsLibrary
 - * transformationProductsFormula
 - transformationProductsAnnForm
 - transformationProductsLibraryFormula
 - * transformationProductsLogic
 - features
 - * featuresSet
 - * featuresUnset
 - * featuresFromFeatGroups
 - * featuresConsensus
 - * featuresBruker
 - * featuresEnviPick
 - * featuresKPIC2
 - * featuresOpenMS
 - * featuresPiek
 - * featuresSAFD
 - * featuresSIRIUS
 - * featuresTable
 - * featuresBrukerTASQ
 - * featuresXCMS
 - * featuresXCMS3
 - featureGroups
 - * featureGroupsSet
 - featureGroupsScreeningSet
 - * featureGroupsUnset
 - * featureGroupsScreening
 - featureGroupsSetScreeningUnset
 - * featureGroupsBruker
 - * featureGroupsConsensus
 - * featureGroupsEnviMass
 - * featureGroupsGreedy
 - * featureGroupsIMS
 - * featureGroupsKPIC2
 - * featureGroupsOpenMS

- * featureGroupsSIRIUS
- * featureGroupsTable
- * featureGroupsBrukerTASQ
- * featureGroupsXCMS
- * featureGroupsXCMS3
- components
 - * componentsCamera
 - * componentsFeatures
 - componentsCliqueMS
 - componentsOpenMS
 - * componentsClust
 - componentsIntClust
 - componentsSpecClust
 - * componentsSet
 - componentsNTSet
 - * componentsUnset
 - * componentsNT
 - componentsNTUnset
 - * componentsRC
 - * componentsTPs
- featureAnnotations
 - * formulas
 - formulasConsensus
 - formulasSet
 - formulasUnset
 - formulasSIRIUS
 - * compounds
 - compoundsConsensus
 - compoundsMF
 - compoundsSet
 - compoundsUnset
 - compoundsSIRIUS
- MSPeakLists
 - * MSPeakListsSet
 - * MSPeakListsUnset
- MSLibrary

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
 - compoundsSet
 - * compoundsConsensusSet
 - formulasSet
 - * formulasConsensusSet
 - MSPeakListsSet

xcms-conv

*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, IMS = FALSE)

## S4 method for signature 'featuresXCMS'
getXCMSSet(obj, verbose = TRUE, ...)

## S4 method for signature 'featureGroups'
getXCMSSet(obj, verbose, loadRawData, IMS = FALSE)

## 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, IMS = FALSE)

## S4 method for signature 'featuresXCMS3'
```



```

getXCMSnExp(obj, verbose = TRUE, ...)

## S4 method for signature 'featureGroups'
getXCMSnExp(obj, verbose, loadRawData, IMS = FALSE)

## 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 (<i>e.g.</i> file paths) is used in the returned object.
IMS	(IMS workflow) Specifies which feature groups are considered for export in IMS workflows. The following options are valid: <ul style="list-style-type: none"> • "both": Selects IMS and non-IMS features. • "maybe": Selects non-IMS features and IMS features without assigned IMS precursor. • FALSE: Selects only non-IMS features. • TRUE: Selects only IMS features. This should be kept FALSE as XCMS export currently does not support IMS features.
set	(sets workflow) The name of the set to be exported.

Details

The conversion process will introduce some dummy values for metadata not present in **patRo** objects. If the features or featureGroups object was generated with **XCMS**, then no conversion is performed and the original **XCMS** object will be returned, if possible. Conversion may still occur *e.g.* due to the application of some subsetting or filtering steps or the re-ordering of analyses.

Sets workflows

In a [sets workflow](#), [unset](#) is used to convert the feature (group) data before the object is exported.

References

reference 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.

Index

[(generics), 229
 [,MSLibrary,ANY,missing,missing-method
 (MSLibrary-class), 307
 [,MSPeakLists,ANY,ANY,missing-method
 (MSPeakLists-class), 313
 [,MSPeakListsSet,ANY,ANY,missing-method
 (MSPeakLists-class), 313
 [,components,ANY,ANY,missing-method
 (components-class), 38
 [,componentsSet,ANY,ANY,missing-method
 (components-class), 38
 [,compoundsCluster,ANY,missing,missing-method
 (compoundsCluster-class), 64
 [,compoundsSet,ANY,missing,missing-method
 (compounds-class), 54
 [,featureAnnotations,ANY,missing,missing-method
 (featureAnnotations-class), 95
 [,featureGroups,ANY,ANY,missing-method
 (featureGroups-class), 101
 [,featureGroupsComparison,ANY,missing,missing-method
 (featureGroupsComparison-class),
 118
 [,featureGroupsScreening,ANY,ANY,missing-method
 (featureGroupsScreening-class),
 120
 [,featureGroupsScreeningSet,ANY,ANY,missing-method
 (featureGroupsScreening-class),
 120
 [,featureGroupsSet,ANY,ANY,missing-method
 (featureGroups-class), 101
 [,features,ANY,missing,missing-method
 (features-class), 124
 [,featuresSet,ANY,missing,missing-method
 (features-class), 124
 [,formulasSet,ANY,missing,missing-method
 (formulas-class), 150
 [,transformationProducts,ANY,missing,missing-method
 (transformationProducts-class),
 351
 [[(generics), 229
 [[,MSLibrary,ANY,missing-method
 (MSLibrary-class), 307
 [[,MSPeakLists,ANY,ANY-method
 (MSPeakLists-class), 313
 [[,components,ANY,ANY-method
 (components-class), 38
 [[,featureAnnotations,ANY,missing-method
 (featureAnnotations-class), 95
 [[,featureGroups,ANY,ANY-method
 (featureGroups-class), 101
 [[,featureGroupsComparison,ANY,missing-method
 (featureGroupsComparison-class),
 118
 [[,features,ANY,missing-method
 (features-class), 124
 [[,formulas,ANY,ANY-method
 (formulas-class), 150
 [[,transformationProducts,ANY,missing-method
 (transformationProducts-class),
 351
 \$(generics), 229
 \$[,MSLibrary-method (MSLibrary-class),
 307
 \$[,MSPeakLists-method
 (MSPeakLists-class), 313
 \$[,components-method (components-class),
 38
 \$[,featureAnnotations-method
 (featureAnnotations-class), 95
 \$[,featureGroups-method
 (featureGroups-class), 101
 \$[,featureGroupsComparison-method
 (featureGroupsComparison-class),
 118
 \$[,features-method (features-class), 124
 \$[,transformationProducts-method
 (transformationProducts-class),
 351

- addAllDAEICs (bruker-utils), 29
- addDAEIC (bruker-utils), 29
- addFormulaScoring, 196
- addFormulaScoring (compounds-class), 54
- addFormulaScoring, compounds-method (compounds-class), 54
- addFormulaScoring, compoundsSet-method (compounds-class), 54
- adduct, 13, 14, 20, 41, 140, 183, 187, 192, 199, 204, 228, 238, 282, 287, 292, 295, 298, 310, 348
- adduct (adduct-class), 12
- adduct utilities, 13
- adduct-class, 12
- adduct-utils, 13
- adducts (generics), 229
- adducts, featureGroups-method (featureGroups-class), 101
- adducts, featureGroupsSet-method (featureGroups-class), 101
- adducts<- (generics), 229
- adducts<-, featureGroups-method (featureGroups-class), 101
- adducts<-, featureGroupsSet-method (featureGroups-class), 101
- algorithm, 117
- algorithm (generics), 229
- algorithm, optimizationResult-method (optimizationResult-class), 323
- algorithm, workflowStep-method (workflowStep-class), 364
- analyses (generics), 229
- analyses, featureGroups-method (featureGroups-class), 101
- analyses, features-method (features-class), 124
- analyses, formulas-method (formulas-class), 150
- analyses, MSPeakLists-method (MSPeakLists-class), 313
- analysis info, 75, 241, 254, 322
- Analysis info table, 30, 79
- analysis info table, 302
- Analysis information, 132, 133, 135–137, 140, 145, 147–149, 250, 260, 272, 274–277, 279–281, 283, 284, 321
- analysis information, 19, 86, 87, 90, 93–95, 107, 108, 111, 114, 128–130, 232, 233, 281, 282, 298, 305, 306, 323, 331
- analysis-information, 15, 76
- analysisInfo, 146, 282
- analysisInfo (generics), 229
- analysisInfo, featureGroups-method (featureGroups-class), 101
- analysisInfo, features-method (features-class), 124
- analysisInfo, MSPeakListsSet-method (MSPeakLists-class), 313
- analysisinfo-dataframe, 17
- analysisInfo<- (generics), 229
- analysisInfo<-, featureGroups-method (featureGroups-class), 101
- analysisInfo<-, featureGroupsXCMS-method (featureGroups-class), 101
- analysisInfo<-, features-method (features-class), 124
- analysisInfo<-, featuresXCMS-method (features-class), 124
- annotatedPeakList (generics), 229
- annotatedPeakList, compounds-method (compounds-class), 54
- annotatedPeakList, compoundsSet-method (compounds-class), 54
- annotatedPeakList, formulas-method (formulas-class), 150
- annotatedPeakList, formulasSet-method (formulas-class), 150
- annotation similarity, 181, 183, 186, 192, 196, 199, 204, 212, 215
- annotations, 267
- annotations (generics), 229
- annotations, featureAnnotations-method (featureAnnotations-class), 95
- annotations, featureGroups-method (featureGroups-class), 101
- annotations, formulas-method (formulas-class), 150
- as.adduct, 13, 14, 20, 41, 140, 183, 187, 192, 199, 204, 228, 287, 292, 295, 296, 310, 348, 349
- as.adduct (adduct-utils), 13
- as.character, adduct-method (adduct-class), 12
- as.data.frame (generics), 229
- as.data.frame, workflowStep-method

- (workflowStep-class), 364
- as.data.table, 17, 114, 165, 179, 267, 275, 276, 281, 282, 326, 331, 335
- as.data.table(generics), 229
- as.data.table, components-method (components-class), 38
- as.data.table, componentsTPs-method (componentsTPs-class), 51
- as.data.table, featureAnnotations-method (featureAnnotations-class), 95
- as.data.table, featureGroups-method (feature-table), 92
- as.data.table, featureGroupsScreening-method (feature-table), 92
- as.data.table, featureGroupsScreeningSet-method (feature-table), 92
- as.data.table, features-method (features-class), 124
- as.data.table, featuresSet-method (features-class), 124
- as.data.table, formulas-method (formulas-class), 150
- as.data.table, MSLibrary-method (MSLibrary-class), 307
- as.data.table, MSPeakLists-method (MSPeakLists-class), 313
- as.data.table, MSPeakListsSet-method (MSPeakLists-class), 313
- as.data.table, transformationProducts-method (transformationProducts-class), 351
- as.data.table, workflowStep-method (workflowStep-class), 364
- assignMobilities, 16, 43, 61, 115, 140, 166, 176, 180, 184, 188, 190, 194, 200, 206, 320, 332
- assignMobilities(generics), 229
- assignMobilities method for suspects, 21
- assignMobilities, compounds-method (assignMobilities_comp), 20
- assignMobilities, compoundsSet-method (assignMobilities_comp), 20
- assignMobilities, data.frame-method (assignMobilities_susp), 26
- assignMobilities, data.table-method (assignMobilities_susp), 26
- assignMobilities, featureGroups-method (assignMobilities_feat), 21
- assignMobilities, featureGroupsScreening-method (assignMobilities_feat), 21
- assignMobilities, featureGroupsScreeningSet-method (assignMobilities_feat), 21
- assignMobilities, featureGroupsSet-method (assignMobilities_feat), 21
- assignMobilities_comp, 20
- assignMobilities_feat, 21
- assignMobilities_susp, 26
- assignMobilities, 161, 164, 168, 172, 174
- availableBackends(msdata), 304
- averagedPeakLists(MSPeakLists-class), 313
- averagedPeakLists, MSPeakLists-method (MSPeakLists-class), 313
- bruker-utils, 29
- C++Object, 238
- C3SDBAdducts, 27, 31
- caching, 32
- calculateConcs, 25, 74, 76, 94, 113
- calculateConcs(pred-quant), 327
- calculateConcs, featureGroups-method (pred-quant), 327
- calculateConcs, featureGroupsScreening-method (pred-quant), 327
- calculateConcs, featureGroupsScreeningSet-method (pred-quant), 327
- calculateConcs, featureGroupsSet-method (pred-quant), 327
- calculateIonFormula(adduct-utils), 13
- calculateJaggedness, 109, 128
- calculateModality, 109, 128
- calculateNeutralFormula(adduct-utils), 13
- calculatePeakQualities, 25, 75, 91, 92, 128
- calculatePeakQualities(generics), 229
- calculatePeakQualities, featureGroups-method (featureGroups-class), 101
- calculatePeakQualities, features-method (features-class), 124
- calculateTox, 25, 75, 76, 94, 113
- calculateTox(pred-tox), 332
- calculateTox, featureGroups-method (pred-tox), 332

- calculateTox, featureGroupsScreening-method
(pred-tox), 332
- calculateTox, featureGroupsScreeningSet-method
(pred-tox), 332
- calculateTox, featureGroupsSet-method
(pred-tox), 332
- CAMERA::annotate, 161
- CAMERA::findIsotopes, 79
- CCS-Conversion, 33
- CentWaveParam, 80
- check-GUI, 34, 41, 76
- checkComponents, 41, 112
- checkComponents (check-GUI), 34
- checkComponents, components-method
(check-GUI), 34
- checkFeatures, 76
- checkFeatures (check-GUI), 34
- checkFeatures, featureGroups-method
(check-GUI), 34
- chordDiagram, 86, 117
- clearCache (caching), 32
- cliqueMS::getAnnotation, 163, 164
- cliqueMS::getCliques, 162, 163
- cliqueMS::getIsotopes, 163, 164
- cluster-params, 37
- clustering parameters, 244, 246, 302
- clusterProperties (generics), 229
- clusterProperties, componentsClust-method
(componentsClust-class), 44
- clusterProperties, compoundsCluster-method
(compoundsCluster-class), 64
- clusters (generics), 229
- clusters, componentsClust-method
(componentsClust-class), 44
- clusters, compoundsCluster-method
(compoundsCluster-class), 64
- comparison, 119
- comparison (featureGroups-compare), 116
- comparison, featureGroups-method
(featureGroups-compare), 116
- comparison, featureGroupsSet-method
(featureGroups-compare), 116
- componentInfo, 178
- componentInfo (components-class), 38
- componentInfo, components-method
(components-class), 38
- componentization, 112
- components, 35, 41, 43, 44, 46, 47, 49–51, 53,
76, 109, 160, 161, 174, 233–238,
337, 342, 366, 367
- components (components-class), 38
- components-class, 38
- componentsCamera, 43, 366
- componentsCamera (components-class), 38
- componentsCamera-class
(components-class), 38
- componentsCliqueMS, 43, 366
- componentsCliqueMS (components-class),
38
- componentsCliqueMS-class
(components-class), 38
- componentsClust, 43, 46–48, 51, 233, 235,
236, 366
- componentsClust
(componentsClust-class), 44
- componentsClust-class, 44
- componentsFeatures, 43, 164, 171, 238, 366
- componentsFeatures (components-class),
38
- componentsFeatures-class
(components-class), 38
- componentsIntClust, 43, 46, 48, 166, 235,
366
- componentsIntClust
(componentsIntClust-class), 47
- componentsIntClust-class, 47
- componentsNT, 43, 168, 235, 366
- componentsNT (componentsNT-class), 49
- componentsNT-class, 49
- componentsNTSet, 43, 168, 235, 236, 366, 368
- componentsNTSet (componentsNT-class), 49
- componentsNTSet-class
(componentsNT-class), 49
- componentsNTUnset, 43, 366
- componentsNTUnset (componentsNT-class),
49
- componentsNTUnset-class
(componentsNT-class), 49
- componentsOpenMS, 43, 366
- componentsOpenMS (components-class), 38
- componentsOpenMS-class
(components-class), 38
- componentsRC, 43, 366
- componentsRC (components-class), 38
- componentsRC-class (components-class),
38

- componentsSet, [43](#), [161](#), [164](#), [172](#), [174](#), [233](#),
[234](#), [236](#), [238](#), [366](#), [368](#)
- componentsSet (components-class), [38](#)
- componentsSet-class (components-class),
[38](#)
- componentsSpecClust, [43](#), [46](#), [51](#), [176](#), [366](#)
- componentsSpecClust
(componentsSpecClust-class), [51](#)
- componentsSpecClust-class, [51](#)
- componentsTPs, [43](#), [53](#), [178](#), [179](#), [234](#), [235](#),
[237](#), [357](#), [360](#), [366](#)
- componentsTPs (componentsTPs-class), [51](#)
- componentsTPs-class, [51](#)
- componentsUnset, [43](#), [366](#)
- componentsUnset (components-class), [38](#)
- componentsUnset-class
(components-class), [38](#)
- componentTable, [179](#)
- componentTable (components-class), [38](#)
- componentTable, components-method
(components-class), [38](#)
- compound annotation, [64](#)
- compound annotation candidates, [210](#)
- compound generation, [208](#)
- compound generators, [60](#)
- compounds, [20](#), [21](#), [58](#), [61](#), [63](#), [64](#), [68–70](#), [100](#),
[101](#), [178](#), [181](#), [182](#), [188](#), [211](#), [217](#),
[219](#), [220](#), [222](#), [223](#), [225](#), [233–236](#),
[238](#), [266](#), [267](#), [318](#), [329](#), [333](#), [337](#),
[342](#), [364](#), [366](#), [367](#)
- compounds (compounds-class), [54](#)
- compounds-class, [54](#)
- compounds-cluster, [63](#)
- compoundsCluster, [64](#), [233–238](#), [337](#), [342](#)
- compoundsCluster
(compoundsCluster-class), [64](#)
- compoundsCluster-class, [64](#)
- compoundsConsensus, [61](#), [101](#), [366](#)
- compoundsConsensus (compounds-class), [54](#)
- compoundsConsensus-class
(compounds-class), [54](#)
- compoundsConsensusSet, [61](#), [236](#), [368](#)
- compoundsConsensusSet
(compounds-class), [54](#)
- compoundsConsensusSet-class
(compounds-class), [54](#)
- compoundScorings, [68](#), [98](#), [181](#), [187](#), [188](#), [190](#)
- compoundsMF, [61](#), [69](#), [101](#), [188](#), [366](#)
- compoundsMF (compoundsMF-class), [68](#)
- compoundsMF-class, [68](#)
- compoundsSet, [61](#), [101](#), [233–236](#), [238](#), [366](#),
[368](#)
- compoundsSet (compounds-class), [54](#)
- compoundsSet-class (compounds-class), [54](#)
- compoundsSIRIUS, [61](#), [70](#), [101](#), [194](#), [236](#), [329](#),
[334](#), [366](#)
- compoundsSIRIUS
(compoundsSIRIUS-class), [69](#)
- compoundsSIRIUS-class, [69](#)
- compoundsUnset, [61](#), [101](#), [366](#)
- compoundsUnset (compounds-class), [54](#)
- compoundsUnset-class (compounds-class),
[54](#)
- Concentration prediction, [335](#)
- concentrations, [330](#)
- concentrations (featureGroups-class),
[101](#)
- concentrations, featureGroups-method
(featureGroups-class), [101](#)
- consensus (generics), [229](#)
- consensus, components-method
(components-class), [38](#)
- consensus, componentsSet-method
(components-class), [38](#)
- consensus, compounds-method
(compounds-class), [54](#)
- consensus, compoundsSet-method
(compounds-class), [54](#)
- consensus, featureGroupsComparison-method
(featureGroups-compare), [116](#)
- consensus, featureGroupsComparisonSet-method
(featureGroups-compare), [116](#)
- consensus, formulas-method
(formulas-class), [150](#)
- consensus, formulasSet-method
(formulas-class), [150](#)
- consensus, transformationProductsStructure-method
(transformationProductsStructure-class),
[358](#)
- contour, [324](#), [325](#)
- conversion of raw MS data, [16](#)
- convertCCSToMobility (CCS-Conversion),
[33](#)
- convertMobilityToCCS (CCS-Conversion),
[33](#)
- convertMSFiles, [133](#), [135](#), [136](#), [138](#),

- 147–149, 260
- convertMSFiles (MSConversion), 299
- convertMSFilesBruker (MSConversion), 299
- convertMSFilesIMSCollapse (MSConversion), 299
- convertMSFilesOpenMS (MSConversion), 299
- convertMSFilesPaths (MSConversion), 299
- convertMSFilesPWiz (MSConversion), 299
- convertMSFilesTIMSCONVERT (MSConversion), 299
- convertToMFDB (generics), 229
- convertToMFDB, transformationProductsStructure-method (transformationProductsStructure-class), 120
- 358
- convertToSuspects, 179
- convertToSuspects (generics), 229
- convertToSuspects, MSLibrary-method (MSLibrary-class), 307
- convertToSuspects, transformationProducts-method (transformationProducts-class), 351
- cutClusters (generics), 229
- cutClusters, componentsClust-method (componentsClust-class), 44
- cutClusters, compoundsCluster-method (compoundsCluster-class), 64
- cutree, 45, 46, 66, 67, 232
- cutreeDynamicTree, 45, 46, 64, 66, 67, 166, 175, 176, 232
- daisy, 46, 165, 166
- data.table, 42, 43, 53, 92, 94, 99, 108, 112, 113, 123, 128, 129, 155, 237, 241, 311, 320, 353, 364
- defaultExclNormScores (generics), 229
- defaultExclNormScores, compounds-method (compounds-class), 54
- defaultExclNormScores, formulas-method (formulas-class), 150
- defaultLim (limits), 288
- defaultOpenMSAdducts, 70, 170
- delete (generics), 229
- delete, components-method (components-class), 38
- delete, componentsClust-method (componentsClust-class), 44
- delete, componentsSet-method (components-class), 38
- delete, compoundsSet-method (compounds-class), 54
- delete, compoundsSIRIUS-method (compounds-class), 54
- delete, featureAnnotations-method (featureAnnotations-class), 95
- delete, featureGroups-method (featureGroups-class), 101
- delete, featureGroupsKPIC2-method (featureGroups-class), 101
- delete, featureGroupsScreening-method (featureGroupsScreening-class), 120
- delete, featureGroupsScreeningSet-method (featureGroupsScreening-class), 120
- delete, featureGroupsSet-method (featureGroups-class), 101
- delete, featureGroupsXCMS-method (featureGroups-class), 101
- delete, featureGroupsXCMS3-method (featureGroups-class), 101
- delete, features-method (features-class), 124
- delete, featuresKPIC2-method (features-class), 124
- delete, featuresPieK-method (features-class), 124
- delete, featuresXCMS-method (features-class), 124
- delete, featuresXCMS3-method (features-class), 124
- delete, formulas-method (formulas-class), 150
- delete, formulasSet-method (formulas-class), 150
- delete, formulasSIRIUS-method (formulas-class), 150
- delete, MSLibrary-method (MSLibrary-class), 307
- delete, MSPeakLists-method (MSPeakLists-class), 313
- delete, MSPeakListsSet-method (MSPeakLists-class), 313
- delete, transformationProducts-method (transformationProducts-class), 351
- detectCores, 10

- digest, [32](#)
- direct mobility assignment, [142](#)
- do.findmain, [173](#), [174](#)
- draw.pairwise.venn, [89](#), [99](#), [117](#), [118](#), [361](#)
- draw.triple.venn, [89](#), [99](#), [118](#), [361](#)
- drop, [317](#)
- dynamicTreeCut, [46](#), [64](#), [67](#)

- EIC parameters, [35](#), [36](#), [42](#), [88](#), [110](#), [129](#),
[287](#), [337](#), [339](#), [343](#)
- EIM parameters, [35](#), [88](#), [337](#)
- EIXParams, [71](#)
- enviPick, [16](#)
- enviPick::mzpick, [247](#), [248](#)
- enviPickwrap, [134](#), [135](#)
- estimateIDConfidence, [123](#), [124](#), [267](#), [349](#)
- estimateIDConfidence(id-conf), [264](#)
- estimateIDConfidence,compounds-method
(id-conf), [264](#)
- estimateIDConfidence,compoundsSet-method
(id-conf), [264](#)
- estimateIDConfidence,featureGroupsScreening-method
(id-conf), [264](#)
- estimateIDConfidence,featureGroupsScreeningSet-method
(id-conf), [264](#)
- estimateIDConfidence,formulas-method
(id-conf), [264](#)
- estimateIDConfidence,formulasSet-method
(id-conf), [264](#)
- executeMultiProcess, [200](#)
- expandForIMS, [46](#), [166](#), [176](#), [180](#)
- expandForIMS(components-class), [38](#)
- expandForIMS,components-method
(components-class), [38](#)
- expandForIMS,componentsCamera-method
(components-class), [38](#)
- expandForIMS,componentsCliqueMS-method
(components-class), [38](#)
- expandForIMS,componentsNT-method
(components-class), [38](#)
- expandForIMS,componentsNTSet-method
(components-class), [38](#)
- expandForIMS,componentsOpenMS-method
(components-class), [38](#)
- expandForIMS,componentsRC-method
(components-class), [38](#)
- expandForIMS,componentsSet-method
(components-class), [38](#)

- experimentInfo
(optimizationResult-class), [323](#)
- experimentInfo,optimizationResult-method
(optimizationResult-class), [323](#)
- export(generics), [229](#)
- export,featureGroups-method
(featureGroups-class), [101](#)
- export,featureGroupsSet-method
(featureGroups-class), [101](#)
- export,MSLibrary-method
(MSLibrary-class), [307](#)

- fastcluster, [272](#)
- fastcluster::hclust, [165](#), [166](#), [175](#), [176](#)
- feature concentrations, [95](#)
- feature consensus, [256](#)
- feature-filtering, [72](#), [116](#)
- feature-optimization, [78](#)
- feature-plotting, [82](#), [116](#), [252](#)
- feature-quality, [91](#)
- feature-table, [92](#), [116](#)
- featureAnnotations, [57–61](#), [63](#), [76](#), [100](#),
[109](#), [153](#), [155](#), [156](#), [158](#), [233–237](#),
[329](#), [333](#), [366](#)
- featureAnnotations
(featureAnnotations-class), [95](#)
- featureAnnotations-class, [95](#)
- featureGroupQualities, [109](#)
- featureGroupQualities
(feature-quality), [91](#)
- featureGroups, [20](#), [23](#), [30](#), [31](#), [35](#), [36](#), [40](#), [41](#),
[61](#), [74](#), [76](#), [77](#), [97](#), [107](#), [113](#),
[117–124](#), [130](#), [156](#), [159](#), [161](#), [163](#),
[165](#), [167](#), [170](#), [173](#), [175](#), [177](#), [178](#),
[181](#), [183](#), [186](#), [192](#), [196](#), [198](#), [204](#),
[208](#), [211](#), [228](#), [232–238](#), [255](#), [256](#),
[258–260](#), [262](#), [263](#), [270](#), [271](#), [273](#),
[274](#), [276–278](#), [298](#), [299](#), [313](#), [317](#),
[325](#), [328–330](#), [333](#), [334](#), [337](#), [339](#),
[342](#), [347](#), [348](#), [364](#), [365](#), [368](#)
- featureGroups(featureGroups-class), [101](#)
- featureGroups documentation, [17](#)
- featureGroups-class, [101](#)
- featureGroups-compare, [116](#)
- featureGroupsBruker, [113](#), [365](#)
- featureGroupsBruker
(featureGroups-class), [101](#)
- featureGroupsBruker-class
(featureGroups-class), [101](#)

- featureGroupsBrukerTASQ, [114](#), [366](#)
- featureGroupsBrukerTASQ
 - (importFeatureGroupsBrukerTASQ), [272](#)
- featureGroupsComparison, [118](#), [233–235](#), [237](#)
- featureGroupsComparison
 - (featureGroupsComparison-class), [118](#)
- featureGroupsComparison-class, [118](#)
- featureGroupsComparisonSet, [233](#)
- featureGroupsComparisonSet
 - (featureGroupsComparison-class), [118](#)
- featureGroupsComparisonSet-class
 - (featureGroupsComparison-class), [118](#)
- featureGroupsConsensus, [113](#), [365](#)
- featureGroupsConsensus
 - (featureGroups-compare), [116](#)
- featureGroupsConsensus-class
 - (featureGroups-compare), [116](#)
- featureGroupsEnviMass, [113](#), [365](#)
- featureGroupsEnviMass
 - (featureGroups-class), [101](#)
- featureGroupsEnviMass-class
 - (featureGroups-class), [101](#)
- featureGroupsGreedy, [113](#), [365](#)
- featureGroupsGreedy
 - (featureGroups-class), [101](#)
- featureGroupsGreedy-class
 - (featureGroups-class), [101](#)
- featureGroupsIMS, [113](#), [365](#)
- featureGroupsIMS (featureGroups-class), [101](#)
- featureGroupsIMS-class
 - (featureGroups-class), [101](#)
- featureGroupsKPIC2, [113](#), [236](#), [365](#)
- featureGroupsKPIC2
 - (featureGroups-class), [101](#)
- featureGroupsKPIC2-class
 - (featureGroups-class), [101](#)
- featureGroupsOpenMS, [113](#), [365](#)
- featureGroupsOpenMS
 - (featureGroups-class), [101](#)
- featureGroupsOpenMS-class
 - (featureGroups-class), [101](#)
- featureGroupsScreening, [113](#), [123](#), [233](#), [234](#), [236–238](#), [329](#), [333](#), [348](#), [365](#)
- featureGroupsScreening
 - (featureGroupsScreening-class), [120](#)
- featureGroupsScreening-class, [120](#)
- featureGroupsScreeningSet, [113](#), [233](#), [234](#), [236–238](#), [365](#)
- featureGroupsScreeningSet
 - (featureGroupsScreening-class), [120](#)
- featureGroupsScreeningSet-class
 - (featureGroupsScreening-class), [120](#)
- featureGroupsSet, [113](#), [124](#), [232–236](#), [238](#), [276](#), [299](#), [365](#)
- featureGroupsSet (featureGroups-class), [101](#)
- featureGroupsSet-class
 - (featureGroups-class), [101](#)
- featureGroupsSetScreeningUnset, [113](#), [123](#), [365](#)
- featureGroupsSetScreeningUnset
 - (featureGroupsScreening-class), [120](#)
- featureGroupsSetScreeningUnset-class
 - (featureGroupsScreening-class), [120](#)
- featureGroupsSIRIUS, [113](#), [366](#)
- featureGroupsSIRIUS
 - (featureGroups-class), [101](#)
- featureGroupsSIRIUS-class
 - (featureGroups-class), [101](#)
- featureGroupsTable, [114](#), [366](#)
- featureGroupsTable
 - (featureGroups-class), [101](#)
- featureGroupsTable-class
 - (featureGroups-class), [101](#)
- featureGroupsUnset, [113](#), [365](#)
- featureGroupsUnset
 - (featureGroups-class), [101](#)
- featureGroupsUnset-class
 - (featureGroups-class), [101](#)
- featureGroupsXCMS, [114](#), [233](#), [236](#), [366](#)
- featureGroupsXCMS
 - (featureGroups-class), [101](#)
- featureGroupsXCMS-class
 - (featureGroups-class), [101](#)
- featureGroupsXCMS3, [114](#), [236](#), [366](#)

- featureGroupsXCMS3
(featureGroups-class), 101
- featureGroupsXCMS3-class
(featureGroups-class), 101
- featureQualities, 109, 129, 130
- featureQualities (feature-quality), 91
- featureQualityNames (feature-quality), 91
- features, 78, 110–112, 128, 130, 132–136, 138, 146–149, 233–238, 254, 256, 257, 259, 261, 263, 271, 273, 279, 280, 282–284, 286, 298, 325, 365, 368
- features (features-class), 124
- features have been found, 255
- features method of this function, 112
- features-class, 124
- featuresBruker, 131, 365
- featuresBruker (features-class), 124
- featuresBruker-class (features-class), 124
- featuresBrukerTASQ, 131, 365
- featuresBrukerTASQ
(importFeatureGroupsBrukerTASQ), 272
- featuresBrukerTASQ-class
(importFeatureGroupsBrukerTASQ), 272
- featuresConsensus, 131, 365
- featuresConsensus
(featureGroups-compare), 116
- featuresConsensus-class
(featureGroups-compare), 116
- featuresEnviPick, 131, 365
- featuresEnviPick (features-class), 124
- featuresEnviPick-class
(features-class), 124
- featuresFromFeatGroups, 131, 365
- featuresFromFeatGroups
(featureGroups-compare), 116
- featuresFromFeatGroups-class
(featureGroups-compare), 116
- featuresKPIC2, 131, 236, 365
- featuresKPIC2 (features-class), 124
- featuresKPIC2-class (features-class), 124
- featuresOpenMS, 131, 365
- featuresOpenMS (features-class), 124
- featuresOpenMS-class (features-class), 124
- featuresPiek, 131, 236, 365
- featuresPiek (features-class), 124
- featuresPiek-class (features-class), 124
- featuresSAFD, 131, 365
- featuresSAFD (features-class), 124
- featuresSAFD-class (features-class), 124
- featuresSet, 130, 234, 236–238, 282, 299, 365
- featuresSet (features-class), 124
- featuresSet-class (features-class), 124
- featuresSIRIUS, 131, 365
- featuresSIRIUS (features-class), 124
- featuresSIRIUS-class (features-class), 124
- featuresTable, 131, 365
- featuresTable (features-class), 124
- featuresTable-class (features-class), 124
- featuresUnset, 131, 365
- featuresUnset (features-class), 124
- featuresUnset-class (features-class), 124
- featuresXCMS, 131, 233, 236, 365
- featuresXCMS (features-class), 124
- featuresXCMS-class (features-class), 124
- featuresXCMS3, 131, 236, 365
- featuresXCMS3 (features-class), 124
- featuresXCMS3-class (features-class), 124
- featureTable, 113
- featureTable (generics), 229
- featureTable, featureGroups-method
(featureGroups-class), 101
- featureTable, featureGroupsSet-method
(featureGroups-class), 101
- featureTable, features-method
(features-class), 124
- filled.contour, 86, 89
- filter, 16, 21, 35, 36, 88, 109, 179, 190, 213, 245
- filter (generics), 229
- filter method, 250, 331, 335
- filter, components-method
(components-class), 38
- filter, componentsSet-method
(components-class), 38

- filter, componentsTPs-method
(componentsTPs-class), 51
- filter, compounds-method
(compounds-class), 54
- filter, compoundsSet-method
(compounds-class), 54
- filter, featureAnnotations-method
(featureAnnotations-class), 95
- filter, featureGroups-method
(feature-filtering), 72
- filter, featureGroupsScreening-method
(featureGroupsScreening-class), 120
- filter, featureGroupsScreeningSet-method
(featureGroupsScreening-class), 120
- filter, featureGroupsSet-method
(feature-filtering), 72
- filter, features-method
(features-class), 124
- filter, featuresSet-method
(features-class), 124
- filter, formulasSet-method
(formulas-class), 150
- filter, MSLibrary-method
(MSLibrary-class), 307
- filter, MSPeakLists-method
(MSPeakLists-class), 313
- filter, MSPeakListsSet-method
(MSPeakLists-class), 313
- filter, transformationProducts-method
(transformationProducts-class), 351
- filter, transformationProductsAnnComp-method
(transformationProductsAnnComp-class), 354
- filter, transformationProductsAnnForm-method
(transformationProductsAnnForm-class), 355
- filter, transformationProductsStructure-method
(transformationProductsStructure-class), 358
- findFeatures, 15, 79, 80, 132, 132, 134–136, 139, 145–147, 149, 150, 255, 279, 282
- findFeaturesBruker, 133, 133, 271
- findFeaturesEnviPick, 133, 134
- findFeaturesKPIC2, 133, 135
- findFeaturesOpenMS, 133, 136, 248
- findFeaturesPiek, 72, 133, 139, 247
- findFeaturesSAFD, 16, 133, 145
- findFeaturesSIRIUS, 133, 147
- findFeaturesXCMS, 133, 148
- findFeaturesXCMS3, 133, 148, 149, 149
- findFGroup (components-class), 38
- findFGroup, components-method
(components-class), 38
- fmcsR: : fmcs, 211, 212
- formula annotation candidates, 214
- formula generation, 208
- formulas, 52, 57, 60, 100, 101, 153, 156, 158, 159, 178, 196, 197, 201, 214, 233–238, 266, 318, 337, 342, 366
- formulas (formulas-class), 150
- formulas-class, 150
- formulasConsensus, 100, 156, 366
- formulasConsensus (formulas-class), 150
- formulasConsensus-class
(formulas-class), 150
- formulasConsensusSet, 156, 236, 368
- formulasConsensusSet (formulas-class), 150
- formulasConsensusSet-class
(formulas-class), 150
- formulaScorings, 98, 158, 197
- formulasSet, 100, 156, 233–236, 238, 366, 368
- formulasSet (formulas-class), 150
- formulasSet-class (formulas-class), 150
- formulasSIRIUS, 100, 156, 158, 205, 206, 236, 329, 333, 366
- formulasSIRIUS (formulasSIRIUS-class), 158
- formulasSIRIUS-class, 158
- formulasUnset, 100, 156, 366
- formulasUnset (formulas-class), 150
- formulasUnset-class (formulas-class), 150
- fp.sim.matrix, 64, 250
- fread, 275, 281
- fromIMS (generics), 229
- fromIMS, featureGroups-method
(featureGroups-class), 101
- fromIMS, features-method
(features-class), 124
- future.apply, 11

- future_lapply, [11](#)
- generateAnalysisInfo, [19](#)
- generateAnalysisInfo
 - (analysis-information), [15](#)
- generateAnalysisInfoFromEnviMass
 - (analysis-information), [15](#)
- generateComponents, [42](#), [44](#), [47](#), [48](#), [50](#), [51](#),
[53](#), [112](#), [159](#), [162](#), [165](#), [166](#), [169](#),
[172](#), [174](#), [176](#), [180](#)
- generateComponents, featureGroups-method
 - (generateComponents), [159](#)
- generateComponentsCAMERA, [160](#), [160](#)
- generateComponentsCAMERA, featureGroups-method
 - (generateComponentsCAMERA), [160](#)
- generateComponentsCAMERA, featureGroupsSet-method
 - (generateComponentsCAMERA), [160](#)
- generateComponentsCliqueMS, [160](#), [162](#)
- generateComponentsCliqueMS, featureGroups-method
 - (generateComponentsCliqueMS),
[162](#)
- generateComponentsCliqueMS, featureGroupsSet-method
 - (generateComponentsCliqueMS),
[162](#)
- generateComponentsIntClust, [43](#), [44](#), [48](#),
[114](#), [160](#), [165](#)
- generateComponentsIntClust, featureGroups-method
 - (generateComponentsIntClust),
[165](#)
- generateComponentsNontarget, [36](#), [49](#), [160](#),
[167](#)
- generateComponentsNontarget, featureGroups-method
 - (generateComponentsNontarget),
[167](#)
- generateComponentsNontarget, featureGroupsSet-method
 - (generateComponentsNontarget),
[167](#)
- generateComponentsOpenMS, [70](#), [160](#), [169](#)
- generateComponentsOpenMS, featureGroups-method
 - (generateComponentsOpenMS), [169](#)
- generateComponentsOpenMS, featureGroupsSet-method
 - (generateComponentsOpenMS), [169](#)
- generateComponentsRAMClustR, [160](#), [172](#)
- generateComponentsRAMClustR, featureGroups-method
 - (generateComponentsRAMClustR),
[172](#)
- generateComponentsRAMClustR, featureGroupsSet-method
 - (generateComponentsRAMClustR),
[172](#)
- generateComponentsSpecClust, [43](#), [44](#), [51](#),
[160](#), [175](#)
- generateComponentsSpecClust, featureGroups-method
 - (generateComponentsSpecClust),
[175](#)
- generateComponentsTPs, [36](#), [43](#), [51](#), [52](#), [160](#),
[177](#), [250](#), [344](#), [360](#)
- generateComponentsTPs, featureGroups-method
 - (generateComponentsTPs), [177](#)
- generateComponentsTPs, featureGroupsSet-method
 - (generateComponentsTPs), [177](#)
- generateCompounds, [11](#), [60–63](#), [99](#), [180](#), [185](#),
[191](#), [195](#), [267](#)
- generateCompounds, featureGroups-method
 - (generateCompounds), [180](#)
- generateCompoundsLibrary, [182](#), [182](#), [292](#),
[294](#), [295](#), [297](#), [311](#)
- generateCompoundsLibrary, featureGroups-method
 - (generateCompoundsLibrary), [182](#)
- generateCompoundsLibrary, featureGroupsSet-method
 - (generateCompoundsLibrary), [182](#)
- generateCompoundsMetFrag, [11](#), [21](#), [58](#), [60](#),
[69](#), [182](#), [185](#), [343](#)
- generateCompoundsMetFrag, featureGroups-method
 - (generateCompoundsMetFrag), [185](#)
- generateCompoundsMetFrag, featureGroupsSet-method
 - (generateCompoundsMetFrag), [185](#)
- generateCompoundsSIRIUS, [11](#), [69](#), [70](#), [182](#),
[188](#), [191](#), [331](#), [334](#)
- generateCompoundsSIRIUS, featureGroups-method
 - (generateCompoundsSIRIUS), [191](#)
- generateCompoundsSIRIUS, featureGroupsSet-method
 - (generateCompoundsSIRIUS), [191](#)
- generate transformation products, [178](#)
- generateFeatureOptPSet
 - (feature-optimization), [78](#)
- generateFGroupsOptPSet
 - (feature-optimization), [78](#)
- generateFormulas, [99](#), [155–158](#), [195](#), [199](#),
[203](#), [205](#), [207](#), [267](#)
- generateFormulas, featureGroups-method
 - (generateFormulas), [195](#)
- generateFormulasDA, [197](#)
- generateFormulasGenForm, [197](#), [197](#)
- generateFormulasGenForm, featureGroups-method
 - (generateFormulasGenForm), [197](#)
- generateFormulasGenForm, featureGroupsSet-method
 - (generateFormulasGenForm), [197](#)

- generateFormulasSIRIUS, [11](#), [158](#), [159](#), [194](#), [197](#), [203](#), [331](#), [334](#)
- generateFormulasSIRIUS, featureGroups-method (generateFormulasSIRIUS), [203](#)
- generateFormulasSIRIUS, featureGroupsSet-method (generateFormulasSIRIUS), [203](#)
- generateMSPeakLists, [146](#), [207](#), [244](#), [319](#), [322](#)
- generateMSPeakLists, featureGroups-method (generateMSPeakLists), [207](#)
- generateMSPeakLists, featureGroupsSet-method (generateMSPeakLists), [207](#)
- generateMSPeakListsDA, [209](#)
- generateMSPeakListsDAFMF, [209](#)
- generateMSPeakListsMzR, [209](#)
- generateTPs, [209](#), [212](#), [213](#), [216](#), [219](#), [222](#), [225](#), [228](#), [229](#), [249](#), [344](#), [354](#), [358](#), [360](#), [362](#)
- generateTPsAnnComp, [179](#), [180](#), [210](#), [210](#), [355](#)
- generateTPsAnnForm, [179](#), [180](#), [210](#), [211](#), [214](#), [356](#)
- generateTPsBioTransformer, [11](#), [210](#), [211](#), [216](#)
- generateTPsCTS, [210](#), [220](#)
- generateTPsLibrary, [210](#), [222](#), [226–228](#)
- generateTPsLibraryFormula, [210](#), [225](#), [238–240](#)
- generateTPsLogic, [52](#), [179](#), [210](#), [228](#), [240](#), [350](#)
- generateTPsLogic, featureGroups-method (generateTPsLogic), [228](#)
- generateTPsLogic, featureGroupsSet-method (generateTPsLogic), [228](#)
- generics, [229](#)
- GenFormAdducts, [291](#), [295](#)
- GenFormAdducts (adduct-utils), [13](#)
- genFormulaTPLibrary, [228](#), [238](#), [350](#)
- genIDLevelRulesFile, [269](#)
- genIDLevelRulesFile (id-conf), [264](#)
- genLimitsFile (limits), [288](#)
- genReportSettingsFile (reporting), [336](#)
- get.fingerprint, [64](#), [250](#)
- get.mcs, [60](#), [67](#)
- getBGMSMSPeaks, [240](#), [318](#)
- getBPCs (generics), [229](#)
- getBPCs, data.frame-method (analysisinfo-dataframe), [17](#)
- getBPCs, featureGroups-method (featureGroups-class), [101](#)
- getBPCs, features-method (features-class), [124](#)
- getCCSParms, [24](#), [27](#), [33](#), [242](#)
- getDACalibrationError (bruker-utils), [29](#)
- getDefaultRetGroupStartingParams, [81](#)
- getDefaultXcmsSetStartingParams, [81](#)
- getDefAvgPListParams, [208](#), [241](#), [244](#), [302](#), [310](#), [312](#)
- getDefEICParams, [23](#), [143](#), [250](#)
- getDefEICParams (EIXParams), [71](#)
- getDefEIMParams, [23](#)
- getDefEIMParams (EIXParams), [71](#)
- getDefFeaturesOptParamRanges, [79](#)
- getDefFeaturesOptParamRanges (feature-optimization), [78](#)
- getDefFGroupsOptParamRanges, [79](#)
- getDefFGroupsOptParamRanges (feature-optimization), [78](#)
- getDefIsolatePrecParams (MSPeakLists-class), [313](#)
- getDefPeakParams, [23](#), [140](#), [142](#), [246](#)
- getDefPredAggrParams (pred-aggr-params), [326](#)
- getDefSpecSimParams (specSimParams), [345](#)
- getDefTPStructParams, [211](#), [217](#), [221](#), [224](#), [249](#)
- getEICs, [250](#)
- getFCParams, [251](#)
- getFeatureQualityNames, [75](#), [128](#)
- getFeatureQualityNames (generics), [229](#)
- getFeatureQualityNames, featureGroups-method (featureGroups-class), [101](#)
- getFeatureQualityNames, features-method (features-class), [124](#)
- getFeatures (generics), [229](#)
- getFeatures, featureGroups-method (featureGroups-class), [101](#)
- getIMSMATCHParams, [24](#), [58](#), [122](#), [252](#), [347](#)
- getIMSRANGEParams, [58](#), [76](#), [127](#), [253](#)
- getLimIMS (limits), [288](#)
- getMCS (generics), [229](#)
- getMCS, compounds-method (compounds-class), [54](#)
- getMCS, compoundsCluster-method (compoundsCluster-class), [64](#)
- getMCTrainData, [112](#)

- getMCTrainData (check-GUI), 34
- getMSConversionFormats (MSConversion), 299
- getMSConversionTypes (MSConversion), 299
- getMSFileFormats, 253
- getMSFileTypes, 253, 254
- getPeakQualityMetrics, 36
- getPIC, 136, 280
- getPIC.kmeans, 136, 280
- getPICSet (kpic2-conv), 286
- getPICSet, features-method (kpic2-conv), 286
- getPICSet, featuresKPIC2-method (kpic2-conv), 286
- getPiekEICParams (findFeaturesPiek), 139
- getQuantCalibFromScreening (pred-quant), 327
- getTICs (generics), 229
- getTICs, data.frame-method (analysisinfo-dataframe), 17
- getTICs, featureGroups-method (featureGroups-class), 101
- getTICs, features-method (features-class), 124
- getXCMSnExp (xcms-conv), 368
- getXCMSnExp, featureGroups-method (xcms-conv), 368
- getXCMSnExp, featureGroupsSet-method (xcms-conv), 368
- getXCMSnExp, featureGroupsXCMS3-method (xcms-conv), 368
- getXCMSnExp, features-method (xcms-conv), 368
- getXCMSnExp, featuresSet-method (xcms-conv), 368
- getXCMSnExp, featuresXCMS3-method (xcms-conv), 368
- getXCMSSet, 161
- getXCMSSet (xcms-conv), 368
- getXCMSSet, featureGroups-method (xcms-conv), 368
- getXCMSSet, featureGroupsSet-method (xcms-conv), 368
- getXCMSSet, featureGroupsXCMS-method (xcms-conv), 368
- getXCMSSet, features-method (xcms-conv), 368
- getXCMSSet, featuresSet-method (xcms-conv), 368
- getXCMSSet, featuresXCMS-method (xcms-conv), 368
- greedy, 24
- greedy grouping, 24
- greedy grouping algorithm, 25
- groupFeatIndex (featureGroups-class), 101
- groupFeatIndex, featureGroups-method (featureGroups-class), 101
- groupFeatures, 77, 79, 80, 90, 116, 132, 254, 256, 258, 260–262, 264, 270, 274, 276, 298
- groupFeatures, data.frame-method (groupFeatures), 254
- groupFeatures, features-method (groupFeatures), 254
- groupFeaturesGreedy, 24, 255, 255
- groupFeaturesGreedy, features-method (groupFeaturesGreedy), 255
- groupFeaturesKPIC2, 255, 257
- groupFeaturesKPIC2, features-method (groupFeaturesKPIC2), 257
- groupFeaturesKPIC2, featuresSet-method (groupFeaturesKPIC2), 257
- groupFeaturesOpenMS, 255, 258
- groupFeaturesOpenMS, features-method (groupFeaturesOpenMS), 258
- groupFeaturesSIRIUS, 255, 260
- groupFeaturesXCMS, 110, 255, 261, 277
- groupFeaturesXCMS, features-method (groupFeaturesXCMS), 261
- groupFeaturesXCMS, featuresSet-method (groupFeaturesXCMS), 261
- groupFeaturesXCMS3, 255, 262, 278
- groupFeaturesXCMS3, features-method (groupFeaturesXCMS3), 262
- groupFeaturesXCMS3, featuresSet-method (groupFeaturesXCMS3), 262
- groupInfo, 46
- groupInfo (featureGroups-class), 101
- groupInfo, featureGroups-method (featureGroups-class), 101
- groupNames (generics), 229
- groupNames, components-method (components-class), 38
- groupNames, compoundsCluster-method (compoundsCluster-class), 64

- groupNames, featureAnnotations-method
(featureAnnotations-class), 95
- groupNames, featureGroups-method
(featureGroups-class), 101
- groupNames, MSPeakLists-method
(MSPeakLists-class), 313
- groupQualities (featureGroups-class),
101
- groupQualities, featureGroups-method
(featureGroups-class), 101
- groupScores (featureGroups-class), 101
- groupScores, featureGroups-method
(featureGroups-class), 101
- groupTable (featureGroups-class), 101
- groupTable, featureGroups-method
(featureGroups-class), 101

- hasIMS (generics), 229
- hasIMS, featureGroups-method
(featureGroups-class), 101
- hasIMS, featureGroupsComparison-method
(featureGroups-compare), 116
- hasIMS, features-method
(features-class), 124
- hclust, 45, 46, 63, 67
- heatmap.2, 47, 48
- heatmaply, 47, 48
- homol.search, 49, 167, 168
- http::RETRY, 220

- id-conf, 264
- identifiers, 188
- identifiers (compounds-class), 54
- identifiers, compounds-method
(compounds-class), 54
- igraph, 49, 53
- image, 325
- importCheckFeaturesSession (check-GUI),
34
- importFeatureGroups, 270, 272–274,
276–278
- importFeatureGroupsBrukerPA, 270, 271
- importFeatureGroupsBrukerTASQ, 270, 272
- importFeatureGroupsEnviMass, 270, 273
- importFeatureGroupsKPIC2, 270, 274
- importFeatureGroupsTable, 270, 275, 282
- importFeatureGroupsXCMS, 270, 276
- importFeatureGroupsXCMS3, 270, 277
- importFeatures, 278, 279, 280, 282–284
- importFeaturesEnviMass, 273, 279, 279
- importFeaturesKPIC2, 279, 280
- importFeaturesTable, 275, 276, 279, 281
- importFeaturesXCMS, 279, 283
- importFeaturesXCMS3, 279, 283, 284
- importing feature data, 24
- IMS workflow, 276, 282
- installC3SDB, 26, 285
- installTIMSCONVERT, 285, 303
- intensity clusters, 43
- internalStandardAssignments
(featureGroups-class), 101
- internalStandardAssignments, featureGroups-method
(featureGroups-class), 101
- internalStandardAssignments, featureGroupsSet-method
(featureGroups-class), 101
- internalStandards
(featureGroups-class), 101
- internalStandards, featureGroups-method
(featureGroups-class), 101
- its documentation, 19, 25, 36, 89, 113, 131,
139, 145, 209, 241, 251, 287, 303,
339, 343

- kplic2-conv, 286
- KPIC::PICset.align, 257
- KPIC::PICset.group, 257, 274

- launchEICGUI, 287
- launchEICGUI, featureGroups-method
(launchEICGUI), 287
- launchEICGUI, features-method
(launchEICGUI), 287
- length (generics), 229
- length, components-method
(components-class), 38
- length, compoundsCluster-method
(compoundsCluster-class), 64
- length, featureAnnotations-method
(featureAnnotations-class), 95
- length, featureGroups-method
(featureGroups-class), 101
- length, featureGroupsComparison-method
(featureGroupsComparison-class),
118
- length, features-method
(features-class), 124
- length, MSLibrary-method
(MSLibrary-class), 307

- length, MSPeakLists-method
(MSPeakLists-class), 313
- length, optimizationResult-method
(optimizationResult-class), 323
- length, transformationProducts-method
(transformationProducts-class), 351
- lengths (generics), 229
- lengths, compoundsCluster-method
(compoundsCluster-class), 64
- lengths, optimizationResult-method
(optimizationResult-class), 323
- limits, 71, 72, 140, 141, 143, 144, 244, 252, 256, 281, 282, 288, 345
- limits YAML file, 11
- lines, 48, 87
- loadCacheData (caching), 32
- loadMSLibrary, 182, 185, 290, 294, 297, 310, 311, 313
- loadMSLibraryMoNAJSON, 291, 291
- loadMSLibraryMSP, 291, 292, 294

- makeFileHash (caching), 32
- makeHash (caching), 32
- makeHCluster, 66
- makeHCluster (compounds-cluster), 63
- makeHCluster, compounds-method
(compounds-cluster), 63
- makeSet, 297, 344, 345
- makeSet, featureGroups-method (makeSet), 297
- makeSet, featureGroupsSet-method
(makeSet), 297
- makeSet, features-method (makeSet), 297
- makeSet, featuresSet-method (makeSet), 297

- max, 326
- mba.surf, 89
- mean, 326
- merge, MSLibrary, MSLibrary-method
(MSLibrary-class), 307
- MetFragAdducts, 291, 295
- MetFragAdducts (adduct-utils), 13
- method for suspects, 20
- min, 326
- MS file conversion, 15, 17
- MS spectral similarity parameters, 52, 179
- MSConversion, 299

- msdata, 141, 145, 146, 242, 304
- MSLibrary, 183, 185, 233, 234, 236–238, 291, 292, 294, 295, 297, 311, 366
- MSLibrary (MSLibrary-class), 307
- MSLibrary-class, 307
- MSPeakLists, 58, 154, 175, 178, 181, 183, 186, 192, 196, 199, 204, 208, 233–238, 266, 316, 317, 322, 337, 343, 366
- MSPeakLists (MSPeakLists-class), 313
- MSPeakLists-class, 313
- MSPeakListsSet, 233–238, 322, 366, 368
- MSPeakListsSet (MSPeakLists-class), 313
- MSPeakListsSet-class
(MSPeakLists-class), 313
- MSPeakListsUnset, 322, 366
- MSPeakListsUnset (MSPeakLists-class), 313
- MSPeakListsUnset-class
(MSPeakLists-class), 313
- mzR::writeMSData, 302, 303

- names (generics), 229
- names, components-method
(components-class), 38
- names, featureGroups-method
(featureGroups-class), 101
- names, featureGroupsComparison-method
(featureGroupsComparison-class), 118
- names, MSLibrary-method
(MSLibrary-class), 307
- names, transformationProducts-method
(transformationProducts-class), 351

- newProject, 323
- normInts, 25, 76, 89
- normInts (featureGroups-class), 101
- normInts, featureGroups-method
(featureGroups-class), 101
- normInts, featureGroupsSet-method
(featureGroups-class), 101
- numericIDLevel (id-conf), 264

- ObiwarParam, 80
- OpenMS, 16
- OpenTIMS backend, 16
- optimizationResult, 80, 232, 237, 238

- optimizationResult
 - (optimizationResult-class), 323
- optimizationResult-class, 323
- optimizedObject
 - (optimizationResult-class), 323
- optimizedObject, optimizationResult-method
 - (optimizationResult-class), 323
- optimizedParameters
 - (optimizationResult-class), 323
- optimizedParameters, optimizationResult-method
 - (optimizationResult-class), 323
- optimizeFeatureFinding, 325
- optimizeFeatureFinding
 - (feature-optimization), 78
- optimizeFeatureGrouping, 325
- optimizeFeatureGrouping
 - (feature-optimization), 78
- optimizeRetGroup, 81
- optimizeXcmsSet, 81
- options, 10
- overlap (featureGroups-class), 101
- overlap, featureGroups-method
 - (featureGroups-class), 101

- p.adjust, 252
- parallel::detectCores, 306
- parent filter method, 355, 356
- parents (transformationProducts-class), 351
- parents, transformationProducts-method
 - (transformationProducts-class), 351
- patRoan (patRoan-package), 10
- patRoan options, 139, 146, 147, 172, 190, 194, 202, 206, 219, 303
- patRoan-package, 10
- patRoan.path.BioTransformer, 217
- peak detection, 10
- peakLists (MSPeakLists-class), 313
- peakLists, MSPeakLists-method
 - (MSPeakLists-class), 313
- persp, 325
- piek, 24
- plot, 19, 40, 42, 45, 48, 58, 59, 66, 86, 87, 128, 129, 153, 154, 317, 319
- plot (generics), 229
- plot, componentsClust, missing-method
 - (componentsClust-class), 44
- plot, compoundsCluster, missing-method
 - (compoundsCluster-class), 64
- plot, featureGroups, missing-method
 - (feature-plotting), 82
- plot, featureGroupsComparison, missing-method
 - (featureGroups-compare), 116
- plot, optimizationResult, missing-method
 - (optimizationResult-class), 323
- plot.dendrogram, 45, 66
- plotBPCs (generics), 229
- plotBPCs, data.frame-method
 - (analysisinfo-dataframe), 17
- plotBPCs, data.table-method
 - (analysisinfo-dataframe), 17
- plotBPCs, featureGroups-method
 - (feature-plotting), 82
- plotBPCs, features-method
 - (features-class), 124
- plotChord (generics), 229
- plotChord, featureGroups-method
 - (feature-plotting), 82
- plotChord, featureGroupsComparison-method
 - (featureGroups-compare), 116
- plotChroms, 273
- plotChroms (generics), 229
- plotChroms, components-method
 - (components-class), 38
- plotChroms, featureGroups-method
 - (feature-plotting), 82
- plotChroms3D (generics), 229
- plotChroms3D, featureGroups-method
 - (feature-plotting), 82
- plotChroms3D, featureGroupsSet-method
 - (feature-plotting), 82
- plotGraph, 339
- plotGraph (generics), 229
- plotGraph, componentsNT-method
 - (componentsNT-class), 49
- plotGraph, componentsNTSet-method
 - (componentsNT-class), 49
- plotGraph, componentsTPs-method
 - (componentsTPs-class), 51
- plotGraph, featureGroups-method
 - (feature-plotting), 82
- plotGraph, featureGroupsSet-method
 - (feature-plotting), 82
- plotGraph, transformationProductsFormula-method
 - (transformationProductsFormula-class),

- 357
- plotGraph, transformationProductsStructure-method
(transformationProductsStructure-class), 358
- plotHeatMap (componentsIntClust-class), 47
- plotHeatMap, componentsIntClust-method
(componentsIntClust-class), 47
- plotInt, 114
- plotInt (generics), 229
- plotInt, componentsIntClust-method
(componentsIntClust-class), 47
- plotInt, featureGroups-method
(feature-plotting), 82
- plotMobilograms (feature-plotting), 82
- plotMobilograms, featureGroups-method
(feature-plotting), 82
- plotScores (generics), 229
- plotScores, compounds-method
(compounds-class), 54
- plotScores, formulas-method
(formulas-class), 150
- plotSilhouettes (generics), 229
- plotSilhouettes, componentsClust-method
(componentsClust-class), 44
- plotSilhouettes, compoundsCluster-method
(compoundsCluster-class), 64
- plotSpectrum (generics), 229
- plotSpectrum, components-method
(components-class), 38
- plotSpectrum, compounds-method
(compounds-class), 54
- plotSpectrum, compoundsSet-method
(compounds-class), 54
- plotSpectrum, formulas-method
(formulas-class), 150
- plotSpectrum, formulasSet-method
(formulas-class), 150
- plotSpectrum, MSPeakLists-method
(MSPeakLists-class), 313
- plotSpectrum, MSPeakListsSet-method
(MSPeakLists-class), 313
- plotStructure (generics), 229
- plotStructure, compounds-method
(compounds-class), 54
- plotStructure, compoundsCluster-method
(compoundsCluster-class), 64
- plotTICs (generics), 229
- plotTICs, data.frame-method
(analysisinfo-dataframe), 17
- plotTICs, data.table-method
(analysisinfo-dataframe), 17
- plotTICs, featureGroups-method
(feature-plotting), 82
- plotTICs, features-method
(features-class), 124
- plotUpSet (generics), 229
- plotUpSet, featureAnnotations-method
(featureAnnotations-class), 95
- plotUpSet, featureGroups-method
(feature-plotting), 82
- plotUpSet, featureGroupsComparison-method
(featureGroups-compare), 116
- plotUpSet, transformationProductsStructure-method
(transformationProductsStructure-class), 358
- plotVenn (generics), 229
- plotVenn, featureAnnotations-method
(featureAnnotations-class), 95
- plotVenn, featureGroups-method
(feature-plotting), 82
- plotVenn, featureGroupsComparison-method
(featureGroups-compare), 116
- plotVenn, transformationProductsStructure-method
(transformationProductsStructure-class), 358
- plotVolcano, 179
- plotVolcano (generics), 229
- plotVolcano, featureGroups-method
(feature-plotting), 82
- post mobility assignment, 142, 256, 282
- pred-aggr-params, 326
- pred-quant, 327
- pred-tox, 332
- predictCheckFeaturesSession, 76, 112
- predictCheckFeaturesSession
(check-GUI), 34
- predicted concentrations, 326
- prediction aggregation parameters, 76, 94
- predictRespFactors, 95, 122, 205
- predictRespFactors (generics), 229
- predictRespFactors, compounds-method
(pred-quant), 327
- predictRespFactors, compoundsSet-method
(pred-quant), 327

- predictRespFactors, compoundsSIRIUS-method
(pred-quant), [327](#)
- predictRespFactors, featureGroupsScreening-method
(pred-quant), [327](#)
- predictRespFactors, featureGroupsScreeningSet-method
(pred-quant), [327](#)
- predictRespFactors, formulasSet-method
(pred-quant), [327](#)
- predictRespFactors, formulasSIRIUS-method
(pred-quant), [327](#)
- predictTox, [95](#), [122](#), [205](#)
- predictTox (generics), [229](#)
- predictTox, compounds-method (pred-tox),
[332](#)
- predictTox, compoundsSet-method
(pred-tox), [332](#)
- predictTox, compoundsSIRIUS-method
(pred-tox), [332](#)
- predictTox, featureGroupsScreening-method
(pred-tox), [332](#)
- predictTox, featureGroupsScreeningSet-method
(pred-tox), [332](#)
- predictTox, formulasSet-method
(pred-tox), [332](#)
- predictTox, formulasSIRIUS-method
(pred-tox), [332](#)
- printPackageOpts, [335](#)
- products
(transformationProducts-class),
[351](#)
- products, transformationProducts-method
(transformationProducts-class),
[351](#)

- ramclustR, [173](#), [174](#)
- raw data interface, [10](#), [19](#), [25](#), [36](#), [89](#), [113](#),
[131](#), [139](#), [145](#), [209](#), [241](#), [251](#), [287](#),
[303](#), [339](#), [343](#)
- rcdk::get.xlogp, [249](#)
- RColorBrewer, [45](#), [66](#)
- recalibrateDAFiles (bruker-utils), [29](#)
- records (MSLibrary-class), [307](#)
- records, MSLibrary-method
(MSLibrary-class), [307](#)
- regular expression, [266](#)
- regular feature grouping algorithms,
[118](#)
- remotes::install_github, [306](#)
- replicates (generics), [229](#)
- replicates, featureGroups-method
(featureGroups-class), [101](#)
- replicates, features-method
(features-class), [124](#)
- replicateSubtract (feature-filtering),
[72](#)
- replicateSubtract, featureGroups-method
(feature-filtering), [72](#)
- report (reporting), [336](#)
- report, featureGroups-method
(reporting), [336](#)
- reportCSV (reporting-legacy), [340](#)
- reportCSV, featureGroups-method
(reporting-legacy), [340](#)
- reporting, [196](#), [336](#), [340](#)
- reporting-legacy, [340](#)
- reportPDF (reporting-legacy), [340](#)
- reportPDF, featureGroups-method
(reporting-legacy), [340](#)
- retDir, [344](#), [354](#)
- retention order calculation, [249](#)
- retention order direction, [52](#), [178](#), [179](#),
[211](#), [224](#), [227](#), [229](#), [239](#)
- retention order directions, [211](#), [249](#)
- reticulate::py_install, [285](#)
- reticulate::use_virtualenv, [27](#), [302](#)
- reticulate::virtualenv_remove, [285](#)
- revertDAAnalyses (bruker-utils), [29](#)

- saveCacheData (caching), [32](#)
- scores (optimizationResult-class), [323](#)
- scores, optimizationResult-method
(optimizationResult-class), [323](#)
- screenInfo, [267](#)
- screenInfo
(featureGroupsScreening-class),
[120](#)
- screenInfo, featureGroupsScreening-method
(featureGroupsScreening-class),
[120](#)
- screenInfo, featureGroupsScreeningSet-method
(featureGroupsScreening-class),
[120](#)
- screenSuspects, [24](#), [28](#), [107](#), [108](#), [114](#), [144](#),
[179](#), [211](#), [214](#), [217](#), [220](#), [223](#), [226](#),
[238](#), [267](#), [310](#), [311](#), [329](#), [330](#), [353](#)
- screenSuspects (suspect-screening), [346](#)
- screenSuspects, featureGroups-method
(suspect-screening), [346](#)

- screenSuspects, featureGroupsScreening-method
 - (suspect-screening), 346
- screenSuspects, featureGroupsScreeningSet-method
 - (suspect-screening), 346
- screenSuspects, featureGroupsSet-method
 - (suspect-screening), 346
- selectIons, 25, 275
- selectIons (featureGroups-class), 101
- selectIons, featureGroups-method
 - (featureGroups-class), 101
- selectIons, featureGroupsSet-method
 - (featureGroups-class), 101
- set package options, 194, 206
- setDAMethod (bruker-utils), 29
- setObjects, 50
- setObjects (generics), 229
- setObjects, workflowStepSet-method
 - (workflowStepSet-class), 367
- sets (generics), 229
- sets workflow, 21, 26, 41, 58, 76, 95, 107, 128, 153, 160, 161, 164, 166, 168, 172, 174, 176, 180, 182, 197, 209, 276, 282, 298, 317, 350, 369
- sets workflows, 44, 50, 62, 115, 118, 123, 131, 157, 236, 258, 262, 263, 322, 367
- sets, featureGroupsSet-method
 - (featureGroups-class), 101
- sets, featuresSet-method
 - (features-class), 124
- sets, workflowStepSet-method
 - (workflowStepSet-class), 367
- sets-workflow, 344
- settings (compoundsMF-class), 68
- settings, compoundsMF-method
 - (compoundsMF-class), 68
- shiny, 34
- show (generics), 229
- show, adduct-method (adduct-class), 12
- show, components-method
 - (components-class), 38
- show, componentsFeatures-method
 - (components-class), 38
- show, componentsSet-method
 - (components-class), 38
- show, compounds-method
 - (compounds-class), 54
- show, compoundsCluster-method
 - (compoundsCluster-class), 64
- show, compoundsSet-method
 - (compounds-class), 54
- show, featureGroups-method
 - (featureGroups-class), 101
- show, featureGroupsScreening-method
 - (featureGroupsScreening-class), 120
- show, featureGroupsScreeningSet-method
 - (featureGroupsScreening-class), 120
- show, featureGroupsSet-method
 - (featureGroups-class), 101
- show, features-method (features-class), 124
- show, featuresSet-method
 - (features-class), 124
- show, formulas-method (formulas-class), 150
- show, formulasSet-method
 - (formulas-class), 150
- show, MSLibrary-method
 - (MSLibrary-class), 307
- show, MSPeakLists-method
 - (MSPeakLists-class), 313
- show, MSPeakListsSet-method
 - (MSPeakLists-class), 313
- show, optimizationResult-method
 - (optimizationResult-class), 323
- show, transformationProducts-method
 - (transformationProducts-class), 351
- show, workflowStep-method
 - (workflowStep-class), 364
- show, workflowStepSet-method
 - (workflowStepSet-class), 367
- showDataAnalysis (bruker-utils), 29
- signal::sgolayfilt, 72
- specSimParams, 322, 345
- spectra (MSLibrary-class), 307
- spectra, MSLibrary-method
 - (MSLibrary-class), 307
- spectral similarity parameters, 59, 154, 175, 178, 181, 183, 186, 192, 196, 199, 204, 266, 319, 337
- spectrumSimilarity, 176
- spectrumSimilarity (MSPeakLists-class), 313

- spectrumSimilarity,MSPeakLists-method
(MSPeakLists-class), [313](#)
- spectrumSimilarity,MSPeakListsSet-method
(MSPeakLists-class), [313](#)
- spectrumSimilarityIMS
(MSPeakLists-class), [313](#)
- spectrumSimilarityIMS,MSPeakLists-method
(MSPeakLists-class), [313](#)
- suspect list, [107](#), [114](#)
- suspect screening, [140](#), [210](#), [214](#), [217](#), [220](#),
[223](#), [226](#), [238](#)
- Suspect screening results, [331](#), [334](#)
- suspect screening workflows, [94](#)
- suspect-screening, [346](#)
- suspects, [267](#)

- t.test, [252](#)
- toxicities, [95](#), [326](#), [334](#)
- toxicities (featureGroups-class), [101](#)
- toxicities,featureGroups-method
(featureGroups-class), [101](#)
- Toxicity prediction, [332](#)
- TP componentization, [358](#), [361](#)
- TP generators, [353](#), [357](#), [361](#)
- TPLogicTransformations, [228](#), [238](#), [350](#)
- TPs, [43](#)
- TPStructParams, [212](#), [224](#), [227](#), [344](#)
- transformationProducts, [53](#), [178](#), [210](#), [229](#),
[232–234](#), [236–238](#), [337](#), [354](#), [357](#),
[358](#), [361](#), [362](#), [365](#)
- transformationProducts
(transformationProducts-class),
[351](#)
- transformationProducts-class, [351](#)
- transformationProductsAnnComp, [210](#), [212](#),
[234](#), [354](#), [355](#), [362](#), [365](#)
- transformationProductsAnnComp
(transformationProductsAnnComp-class),[354](#)
- transformationProductsAnnComp-class,
[354](#)
- transformationProductsAnnForm, [210](#), [215](#),
[234](#), [354](#), [356](#), [358](#), [365](#)
- transformationProductsAnnForm
(transformationProductsAnnForm-class),[355](#)
- transformationProductsAnnForm-class,
[355](#)
- transformationProductsBT, [354](#), [362](#), [365](#)

- transformationProductsBT
(transformationProductsStructure-class),
[358](#)
- transformationProductsBT-class
(transformationProductsStructure-class),
[358](#)
- transformationProductsCTS, [354](#), [362](#), [365](#)
- transformationProductsCTS
(transformationProductsStructure-class),
[358](#)
- transformationProductsCTS-class
(transformationProductsStructure-class),
[358](#)
- transformationProductsFormula, [210](#), [226](#),
[235](#), [354](#), [356](#), [358](#), [365](#)
- transformationProductsFormula
(transformationProductsFormula-class),
[357](#)
- transformationProductsFormula-class,
[357](#)
- transformationProductsLibrary, [354](#), [362](#),
[365](#)
- transformationProductsLibrary
(transformationProductsStructure-class),
[358](#)
- transformationProductsLibrary-class
(transformationProductsStructure-class),
[358](#)
- transformationProductsLibraryFormula,
[354](#), [358](#), [365](#)
- transformationProductsLibraryFormula
(transformationProductsFormula-class),
[357](#)
- transformationProductsLibraryFormula-class
(transformationProductsFormula-class),
[357](#)
- transformationProductsLogic, [354](#), [365](#)
- transformationProductsLogic
(transformationProducts-class),
[351](#)
- transformationProductsLogic-class
(transformationProducts-class),
[351](#)
- transformationProductsStructure, [210](#),
[211](#), [218](#), [221](#), [224](#), [233–235](#), [354](#),
[355](#), [362](#), [365](#)
- transformationProductsStructure
(transformationProductsStructure-class),

- 358
- transformationProductsStructure-class, 358
- transformationProductsStructureConsensus, 354, 362, 365
- transformationProductsStructureConsensus (transformationProductsStructure-class), 358
- transformationProductsStructureConsensus-class (transformationProductsStructure-class), 358
- treeCut, 46
- treeCut (generics), 229
- treeCut, componentsClust-method (componentsClust-class), 44
- treeCut, compoundsCluster-method (compoundsCluster-class), 64
- treeCutDynamic, 46
- treeCutDynamic (generics), 229
- treeCutDynamic, componentsClust-method (componentsClust-class), 44
- treeCutDynamic, compoundsCluster-method (compoundsCluster-class), 64
- unique (featureGroups-class), 101
- unique, featureGroups-method (featureGroups-class), 101
- unset, 369
- unset (generics), 229
- unset, componentsNTSet-method (componentsNT-class), 49
- unset, componentsSet-method (components-class), 38
- unset, compoundsConsensusSet-method (compounds-class), 54
- unset, compoundsSet-method (compounds-class), 54
- unset, featureGroupsScreeningSet-method (featureGroupsScreening-class), 120
- unset, featureGroupsSet-method (featureGroups-class), 101
- unset, featuresSet-method (features-class), 124
- unset, formulasConsensusSet-method (formulas-class), 150
- unset, formulasSet-method (formulas-class), 150
- unset, MSPeakListsSet-method (MSPeakLists-class), 313
- updateGroups (featureGroups-class), 101
- updateGroups, featureGroups-method (featureGroups-class), 101
- upset, 86, 88, 89, 99, 100, 360, 362
- verifyDependencies, 11, 363
- visNetwork, 49, 52, 53, 88, 89, 357, 358, 360, 361
- withOpt, 363
- withr::with_options, 363
- workflowStep, 43, 100, 113, 130, 232, 237, 238, 311, 322, 354, 365
- workflowStep (workflowStep-class), 364
- workflowStep-class, 364
- workflowStepSet, 44, 50, 62, 124, 157, 236, 238, 322, 345, 368
- workflowStepSet (workflowStepSet-class), 367
- workflowStepSet-class, 367
- xcms-conv, 368
- xcms::adjustRtime, 263
- xcms::findChromPeaks, 149
- xcms::findPeaks, 148
- xcms::group, 261, 262
- xcms::groupChromPeaks, 263
- xcms::peaksWithCentWave, 247, 248
- xcms::retcor, 261, 262
- XCMSnExp, 277, 284, 368
- xcmsSet, 148, 161, 261, 276, 283, 368