Package 'CompoundDb'

April 3, 2026

Type Package

Title Creating and Using (Chemical) Compound Annotation Databases

Version 1.15.4

Description CompoundDb provides functionality to create and use (chemical) compound annotation databases from a variety of different sources such as LipidMaps, HMDB, ChEBI or MassBank. The database format allows to store in addition MS/MS spectra along with compound information. The package provides also a backend for Bioconductor's Spectra package and allows thus to match experimetal MS/MS spectra against MS/MS spectra in the database. Databases can be stored in SQLite format and are thus portable.

Depends R (>= 4.1), methods, AnnotationFilter, S4Vectors

Imports BiocGenerics, ChemmineR, tibble, jsonlite, dplyr, DBI, dbplyr, RSQLite, Biobase, ProtGenerics (>= 1.35.3), xml2, IRanges, Spectra (>= 1.15.10), MsCoreUtils, MetaboCoreUtils, BiocParallel, stringi, data.table

Suggests knitr, rmarkdown, testthat, BiocStyle (>= 2.5.19), MsBackendMgf

URL https://github.com/RforMassSpectrometry/CompoundDb

BugReports https://github.com/RforMassSpectrometry/CompoundDb/issues

biocViews MassSpectrometry, Metabolomics, Annotation

VignetteBuilder knitr

License Artistic-2.0

RoxygenNote 7.3.3

Roxygen list(markdown=TRUE)

Encoding UTF-8

Collate 'AllGenerics.R' 'AnnotationFilters.R' 'createCompDbPackage.R' 'CompDb.R' 'CompDb-methods.R' 'IonDb.R' 'IonDb-methods.R' 'MsBackendCompDb-functions.R' 'MsBackendCompDb.R' 'query-engine.R' 'spectrum-import-functions.R' 'utility-functions.R'

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

git_branch devel

git_last_commit 7daa575

git_last_commit_date 2026-03-19

Repository Bioconductor 3.23

Date/Publication 2026-04-03

Author Jan Stanstrup [aut] (ORCID: <https://orcid.org/0000-0003-0541-7369>),
Johannes Rainer [aut, cre] (ORCID:
<https://orcid.org/0000-0002-6977-7147>),
Josep M. Badia [ctb] (ORCID: <https://orcid.org/0000-0002-5704-1124>),
Roger Gine [aut] (ORCID: <https://orcid.org/0000-0003-0288-9619>),
Andrea Vicini [aut] (ORCID: <https://orcid.org/0000-0001-9438-6909>),
Prateek Arora [ctb] (ORCID: <https://orcid.org/0000-0003-0822-9240>)

Maintainer Johannes Rainer <johannes.rainer@eurac.edu>

Contents

`addJoinDefinition` *Expand a CompDb database with additional, related tables*

Description

The `CompDb` object uses a simple relational database model that consists of the following database tables, some of which are optional:

- *ms_compound*: annotation(s) of compounds.
- *metadata*: general metadata information on the database. This database table is **not** related to any other table in the database and its content is thus also not joined with other database tables.

- *synonym* (optional): database table containing optional additional synonym(s) for compounds in the *ms_compound* table. Rows in this table are linked to a row in *ms_compound* through the "compound_id" database table column.
- *msms_spectrum* (optional): database table with information on individual mass spectra (each row containing the metadata for one spectrum). Database table column "compound_id" links entries in this database table to a single row in the *ms_compound* table.
- *msms_spectrum_peak* (otional): database table containing mass peak data. Each row in this table is related to one row in the *msms_spectrum* table (through the "spectrum_id" column present in both tables).

In addition, the CompDb database layout can be extended by adding additional tables. To make their content automatically available through the built-in compounds() or Spectra() functions, the information on how to combine/join these tables with the existing ones needs to be provided. This can be done using the addJoinDefinition() function: the relationship of a new table with one of the existing tables can be defined with this function providing the names of the two database tables as well as the names of the columns containing the primary/foreign keys defining the the relationship.

See the section *Extending CompDb databases* in the *Creating CompoundDb annotation resources* package vignette for a detailed example.

Usage

```
addJoinDefinition(
  x,
  table_a = character(),
  table_b = character(),
  column_a = character(),
  column_b = character(),
  join = "left outer join"
)
```

Arguments

x	CompDb to which the join definition should be added.
table_a	character(1) with the name of one of the two tables that are related to each other (and can be joined).
table_b	character(1) with the name of the second of the two tables that are related to each other (and can be joined).
column_a	character(1) with the name of the column in table_a containing the keys for the relationship to table table_b.
column_b	character(1) with the name of the column in table_b containing the keys for the relationship to table table_a.
join	character(1) with the type of join. Defaults to join = "left outer join".

Value

The input CompDb with tha added information on how to join the respective database tables.

Author(s)

Johannes Rainer

Examples

```
## The pre-defined table join definitions:
CompoundDb::::.JOINS

## See section "Extending CompDb databases" in the *Creating CompoundDb
## annotation resources* package vignette for examples
```

CompDb *Simple compound (metabolite) databases*

Description

CompDb objects provide access to general (metabolite) compound annotations along with *metadata* information such as the annotation's source, date and release version. The data is stored internally in a database (usually an SQLite database).

hasMsMsSpectra returns TRUE if MS/MS spectrum data is available in the database and FALSE otherwise.

Usage

```
CompDb(x, flags = SQLITE_RO)

hasMsMsSpectra(x)

src_compdb(x)

tables(x)

copyCompDb(x, y)

## S4 method for signature 'CompDb'
dbconn(x)

## S4 method for signature 'CompDb'
Spectra(object, filter, ...)

## S4 method for signature 'CompDb'
supportedFilters(object)

## S4 method for signature 'CompDb'
metadata(x, ...)
```

```
## S4 method for signature 'CompDb'
spectraVariables(object, ...)

## S4 method for signature 'CompDb'
compoundVariables(object, includeId = FALSE, ...)

## S4 method for signature 'CompDb'
compounds(
  object,
  columns = compoundVariables(object),
  filter,
  return.type = c("data.frame", "tibble"),
  ...
)

## S4 method for signature 'CompDb,Spectra'
insertSpectra(object, spectra, columns = spectraVariables(spectra), ...)

## S4 method for signature 'CompDb'
deleteSpectra(object, ids = integer(0), ...)

## S4 method for signature 'CompDb'
mass2mz(x, adduct = c("[M+H]+"), name = "formula")

## S4 method for signature 'CompDb'
insertCompound(object, compounds = data.frame(), addColumns = FALSE)

## S4 method for signature 'CompDb'
deleteCompound(object, ids = character(), recursive = FALSE, ...)
```

Arguments

x	For `CompDb()`: `character(1)` with the file name of the SQLite compound database. Alternatively it is possible to provide the connection to the database with parameter `x`. For `copyCompDb()`: either a `CompDb` or a database connection. `For all other methods: a `CompDb` object.`
flags	flags passed to the SQLite database connection. See `RSQLite::SQLite()`. Defaults to read-only, i.e. `RSQLite::SQLITE_RO`.
y	For `copyCompDb()`: connection to a database to which the content should be copied.
object	For all methods: a `CompDb` object.
filter	For `compounds()` and `Spectra()`: filter expression or `AnnotationFilter::AnnotationFilter()` defining a filter to be used to retrieve specific elements from the database.
...	additional arguments. Currently not used.
includeId	for `compoundVariables()`: `logical(1)` whether the comound ID (column `"compound_id"`) should be included in the result. The default is `includeIds = FALSE`.

columns — For compounds(), Spectra: character with the names of the database columns that should be retrieved. Use compoundVariables() and/or spectraVariables() for a list of available column names. For insertSpectra(): columns (spectra variables) that should be inserted into the database (to avoid inserting all variables).

return.type — For compounds(): either "data.frame" or "tibble" to return the result as a data.frame() or tibble::tibble(), respectively.

spectra — For insertSpectra(): Spectra object containing the spectra to be added to the IonDb database.

ids — For deleteSpectra(): integer() specifying the IDs of the spectra to delete. IDs in ids that are not associated to any spectra in the CompDb object are ignored. For deleteCompound: character() with the compound IDs to be deleted.

adduct — either a character specifying the name(s) of the adduct(s) for which the m/z should be calculated or a data.frame with the adduct definition. See adductNames() for supported adduct names and the description for more information on the expected format if a data.frame is provided.

name — For mass2mz(): character(1). Defines the CompDb column that will be used to name/identify the returned m/z values. By default (name = "formula") m/z values for all unique molecular formulas are calculated and these are used as rownames for the returned matrix. With name = "compound_id" the adduct m/z for all compounds (even those with equal formulas) are calculated and returned.

compounds — For insertCompound(): data.frame with compound data to be inserted into a CompDb database. See function description for details.

addColumns — For insertCompound(): logical(1) whether all (extra) columns in parameter compounds should be stored also in the database table. The default is addColumns = FALSE.

recursive — For deleteCompound(): logical(1) whether also MS2 spectra associated with the compounds should be deleted.

Details

CompDb objects should be created using the constructor function CompDb() providing the name of the (SQLite) database file providing the compound annotation data.

Value

See description of the respective function.

Retrieve annotations from the database

Annotations/compound informations can be retrieved from a CompDb database with the compounds() and Spectra() functions:

- compounds() extracts compound data from the CompDb object. In contrast to src_compdb it returns the actual data as a data.frame (if return.type = "data.frame") or a tibble::tibble() (if return.type = "tibble"). A compounds() call will always return all elements from the *ms_compound* table (unless a filter is used).

- `Spectra()` extract spectra from the database and returns them as a `Spectra::Spectra()` object from the *Spectra* package. Additional annotations requested with the `columns` parameter are added as additional spectra variables.

General functions

- `CompDb()`: connect to a compound database.
- `compoundVariables()`: returns all available columns/database fields for compounds.
- `copyCompDb()`: allows to copy the content from a CompDb to another database. Parameter `x` is supposed to be either a `CompDb` or a database connection from which the data should be copied and `y` a connection to a database to which it should be copied.
- `dbconn()`: returns the connection (of type `DBIConnection`) to the database.
- `metadata()`: returns general meta data of the compound database.
- `spectraVariables()`: returns all spectra variables (i.e. columns) available in the `CompDb`.
- `src_compdb()` provides access to the `CompDb`'s database *via* the functionality from the `dplyr`/`dbplyr` package.
- `supportedFilters()`: provides an overview of the filters that can be applied on a `CompDb` object to extract only specific data from the database.
- `tables()`: returns a named `list` (names being table names) with the fields/columns from each table in the database.
- `mass2mz()`: calculates a table of the m/z values for each compound based on the provided set of adduct(s). Adduct definitions can be provided with parameter `adduct`. See `MetaboCoreUtils::mass2mz()` for more details. Parameter `name` defines the database table column that should be used as `rownames` of the returned `matrix`. By default `name = "formula"`, m/z values are calculated for each unique formula in the `CompDb` `x`.

Adding and removing data from a database

Note that inserting and deleting data requires read-write access to the database. Databases returned by `CompDb` are by default *read-only*. To get write access `CompDb` should be called with parameter `flags = RSQLite::SQLITE_RW`.

- `insertCompound()`: adds additional compound(s) to a `CompDb`. The compound(s) to be added can be specified with parameter `compounds` that is expected to be a `data.frame` with columns `"compound_id"`, `"name"`, `"inchi"`, `"inchikey"`, `"formula"`, `"exactmass"`. Column `"exactmass"` is expected to contain numeric values, all other columns `character`. Missing values are allowed for all columns except `"compound_id"`. An optional column `"synonyms"` can be used to provide alternative names for the compound. This column can contain a single `character` by row, or a `list` with multiple `character` (names) per row/compound (see examples below for details). By setting parameter `addColumns = TRUE` any additional columns in `compound` will be added to the database table. The default is `addColumns = FALSE`. The function returns the `CompDb` with the compounds added. See also `createCompDb()` for more information and details on expected compound data and the examples below for general usage.
- `deleteCompound()`: removes specified compounds from the `CompDb` database. The IDs of the compounds that should be deleted need to be provided with parameter `ids`. To include

compound IDs in the output of a `compounds()` call `"compound_id"` should be added to the `columns` parameter. By default an error is thrown if for some of the specified compounds also MS2 spectra are present in the database. To force deletion of the compounds along with all associated MS2 spectra use `recursive = TRUE`. See examples below for details. The function returns the updated `CompDb` database.

- `insertSpectra()`: adds further spectra to the database. The method always adds all the spectra specified through the `spectra` parameter and does not check if they are already in the database. Note that the input spectra must have the variable `compound_id` and only `Spectra` whose `compound_id` values are also in `compounds(object, "compound_id")` can be added. Parameter `columns` defines which spectra variables from the `spectra` should be inserted into the database. By default, all spectra variables are added but it is strongly suggested to specifically select (meaningful) spectra variables that should be stored in the database. Note that a spectra variable `"compound_id"` is mandatory. If needed, the function adds additional columns to the `msms_spectrum` database table. The function returns the updated `CompDb` object.
- `deleteSpectra()`: deletes specified spectra from the database. The IDs of the spectra to be deleted need to be provided with parameter `ids`.

Note that it is also possible to add new database tables and include them in the data retrieval queries. See `addJoinDefinition()` for more information.

Filtering the database

Data access methods such as `compounds()` and `Spectra` allow to filter the results using specific filter classes and expressions. Filtering uses the concepts from Bioconductor's `AnnotationFilter` package. All information for a certain compound with the ID `"HMDB0000001"` can for example be retrieved by passing the filter expression `filter = ~ compound_id == "HMDB0000001"` to the `compounds` function.

Use the `AnnotationFilter::supportedFilters()` function on the CompDb object to get a list of all supported filters. See also examples below or the usage vignette for details.

Author(s)

Johannes Rainer

See Also

- `createCompDb()` for the function to create a SQLite compound database.
- `CompoundIdFilter()` for filters that can be used on the `CompDb` database.
- `addJoinDefinition()` to expand a *CompDb* with additional, related, database tables.

Examples

```
## We load a small compound test database based on MassBank which is
## distributed with this package.
cdb <- CompDb(system.file("sql/CompDb.MassBank.sql", package = "CompoundDb"))
cdb

## Get general metadata information from the database, such as originating
```

```
## source and version:
metadata(cdb)

## List all available compound annotations/fields
compoundVariables(cdb)

## Extract a data.frame with these annotations for all compounds
compounds(cdb)

## Note that the `compounds` function will by default always return a
## data frame of **unique** entries for the specified columns. Including
## also the `"compound_id"` to the requested columns will ensure that all
## data is returned from the tables.
compounds(cdb, columns = c("compound_id", compoundVariables(cdb)))

## Add also the synonyms (aliases) for the compounds. This will cause the
## tables compound and synonym to be joined. The elements of the compound_id
## and name are now no longer unique
res <- compounds(cdb, columns = c("name", "synonym"))
head(res)

## List all database tables and their columns
tables(cdb)

## Any of these columns can be used in the `compounds` call to retrieve
## the specific annotations. The corresponding database tables will then be
## joined together
compounds(cdb, columns = c("formula", "publication"))

## Calculating m/z values for the exact masses of unique chemical formulas
## in the database:
mass2mz(cdb, adduct = c("[M+H]+", "[M+Na]+"))

## By using `name = "compound_id"` the calculation will be performed for
## each unique compound ID instead (resulting in potentially redundant
## results)
mass2mz(cdb, adduct = c("[M+H]+", "[M+Na]+"), name = "compound_id")

## Create a Spectra object with all MS/MS spectra from the database.
library(Spectra)
sps <- Spectra(cdb)
sps

## Extract spectra for a specific compound.
sps <- Spectra(cdb, filter = ~ name == "Mellein")
sps

## List all available annotations for MS/MS spectra
spectraVariables(sps)

## Get access to the m/z values of these
mz(sps)
```

```
library(Spectra)
## Plot the first spectrum
plotSpectra(sps[1])


#########
## Filtering the database
##
## Get all compounds with an exact mass between 310 and 320
res <- compounds(cdb, filter = ~ exactmass > 310 & exactmass < 320)
res

## Get all compounds that have an H14 in their formula.
res <- compounds(cdb, filter = FormulaFilter("H14", "contains"))
res

#########
## Using CompDb with the *tidyverse*
##
## Using return.type = "tibble" the result will be returned as a "tibble"
compounds(cdb, return.type = "tibble")

## Use the CompDb in a dplyr setup
library(dplyr)
src_cmp <- src_compdb(cdb)
src_cmp

## Get a tbl for the ms_compound table
cmp_tbl <- tbl(src_cmp, "ms_compound")

## Extract the id, name and inchi
cmp_tbl %>% select(compound_id, name, inchi) %>% collect()

########
## Creating an empty CompDb and sequentially adding content
##
## Create an empty CompDb and store the database in a temporary file
cdb <- emptyCompDb(tempfile())
cdb

## Define a data.frame with some compounds to add
cmp <- data.frame(
    compound_id = c(1, 2),
    name = c("Caffeine", "Glucose"),
    formula = c("C8H10N4O2", "C6H12O6"),
    exactmass = c(194.080375584, 180.063388116))

## We can also add multiple synonyms for each compound
cmp$synonyms <- list(c("Cafeina", "Koffein"), "D Glucose")
cmp

## These compounds can be added to the empty database with insertCompound
cdb <- insertCompound(cdb, compounds = cmp)
```

```
compounds(cdb)

## insertCompound would also allow to add additional columns/annotations to
## the database. Below we define a new compound adding an additional column
## hmdb_id
cmp <- data.frame(
    compound_id = 3,
    name = "Alpha-Lactose",
    formula = "C12H22O11",
    exactmass = 342.116211546,
    hmdb_id = "HMDB0000186")

## To add additional columns we need to set addColumns = TRUE
cdb <- insertCompound(cdb, compounds = cmp, addColumns = TRUE)
cdb
compounds(cdb)

######
## Deleting selected compounds from a database
##
## Compounds can be deleted with the deleteCompound function providing the
## IDs of the compounds that should be deleted. IDs of compounds in the
## database can be retrieved by adding "compound_id" to the columns parameter
## of the compounds function:
compounds(cdb, columns = c("compound_id", "name"))

## Compounds can be deleted with the deleteCompound function. Below we delete
## the compounds with the IDs "1" and "3" from the database
cdb <- deleteCompound(cdb, ids = c("1", "3"))
compounds(cdb)

## If also MS2 spectra associated with any of these two compounds an error
## would be thrown. Setting the parameter `recursive = TRUE` in the
## `deleteCompound` call would delete the compounds along with their MS2
## spectra.
```

`compound_tbl_lipidblast` *Extract compound data from LipidBlast*

Description

`compound_tbl_lipidblast()` extracts basic compound annotations from a LipidBlast file in (json format) downloaded from http://mona.fiehnlab.ucdavis.edu/downloads . Note that no mass spectra data is extracted from the json file.

Usage

```
compound_tbl_lipidblast(
  file,
```

```
  collapse = character(),
  n = -1L,
  verbose = FALSE,
  BPPARAM = bpparam()
)
```

Arguments

file	character(1) with the name of the file name.
collapse	optional character(1) to be used to collapse multiple values in the columns "synonyms". See examples for details.
n	integer(1) defining the number of rows from the json file that should be read and processed at a time. By default (n = -1L) the complete file is imported and processed. For large json files it is suggested to set e.g. n = 100000 to enable chunk-wise processing and hence reduce the memory demand.
verbose	logical(1) whether some progress information should be provided. Defaults to verbose = FALSE, but for parsing very large files (specifically with chunk-wise processing enabled with n > 0) it might be helpful to set to verbose = TRUE.
BPPARAM	BiocParallelParam object to configure parallel processing. Defaults to bpparam().

Value

A tibble::tibble with general compound information (one row per compound):

- compound_id: the ID of the compound.
- name: the compound's name.
- inchi: the InChI of the compound.
- inchikey: the InChI key.
- smiles: the SMILES representation of the compound.
- formula: the chemical formula.
- exactmass: the compound's mass.
- compound_class: the class of the compound.
- ionization_mode: the ionization mode.
- precursor_mz: the precursor m/z value.
- precursor_type: the precursor type.
- retention_time: the retention time.
- ccs: the collision cross-section.
- spectrum: the spectrum data (i.e. the mass peaks, as a concatenated character string).
- synonyms: the compound's synonyms (aliases). This type of this column is by default a list to support multiple aliases per compound, unless argument collapse is provided, in which case multiple synonyms are pasted into a single element separated by the value of collapse.

Author(s)

Johannes Rainer, Jan Stanstrup and Prateek Arora

See Also

Other compound table creation functions: `compound_tbl_sdf()`

Examples

```
## Read compound information from a subset of HMDB
fl <- system.file("json/MoNa-LipidBlast_sub.json", package = "CompoundDb")
cmps <- compound_tbl_lipidblast(fl, n = 50000, verbose = TRUE)
cmps
```

---

compound_tbl_sdf *Extract compound data from a file in SDF format*

Description

`compound_tbl_sdf()` extracts basic compound annotations from a file in SDF format (structure-data file). The function currently supports SDF files from:

- HMDB (Human Metabolome Database): http://www.hmdb.ca
- ChEBI (Chemical Entities of Biological Interest): http://ebi.ac.uk/chebi
- LMSD (LIPID MAPS Structure Database): http://www.lipidmaps.org
- PubChem: https://pubchem.ncbi.nlm.nih.gov/
- MoNa: http://mona.fiehnlab.ucdavis.edu/ (see notes below!)

Usage

```
compound_tbl_sdf(file, collapse, onlyValid = TRUE, nonStop = TRUE)
```

Arguments

`file`	`character(1)` with the name of the SDF file.
`collapse`	optional `character(1)` to be used to collapse multiple values in the columns `"synonyms"`. See examples for details.
`onlyValid`	`logical(1)` to import only valid or all elements (defaults to `onlyValid = TRUE`)
`nonStop`	`logical(1)` whether file content specific errors should only reported as warnings and not break the full import process. The value of this parameter is passed to parameter `skipErrors` of the `ChemmineR::read.SDFset()` function.

Details

Column "name" reports for HMDB files the "GENERIC_NAME", for ChEBI the "ChEBI Name", for PubChem the "PUBCHEM_IUPAC_TRADITIONAL_NAME", and for Lipid Maps the "COMMON_NAME", if that is not available, the first of the compounds synonyms and, if that is also not provided, the "SYSTEMATIC_NAME".

Value

A tibble::tibble with general compound information (one row per compound):

- compound_id: the ID of the compound.
- name: the compound's name.
- inchi: the InChI of the compound.
- inchikey: the InChI key.
- formula: the chemical formula.
- exactmass: the compound's (monoisotopic exact) mass.
- synonyms: the compound's synonyms (aliases). This type of this column is by default a list to support multiple aliases per compound, unless argument collapse is provided, in which case multiple synonyms are pasted into a single element separated by the value of collapse.
- smiles: the compound's SMILES (if provided).

Note

compound_tbl_sdf() supports also to read/process gzipped files.

MoNa SDF files organize the data by individual spectra (i.e. each element is one spectrum) and individual compounds can not easily and consistently defined (i.e. not all entries have an InChI ID or other means to uniquely identify compounds). Thus, the function returns a highly redundant compound table. Feedback on how to reduce this redundancy would be highly welcome!

LIPID MAPS was tested August 2020. Older SDF files might not work as the field names were changed.

Author(s)

Johannes Rainer and Jan Stanstrup

See Also

createCompDb() for a function to create a SQLite-based compound database.

Other compound table creation functions: compound_tbl_lipidblast()

Examples

```
## Read compound information from a subset of HMDB
fl <- system.file("sdf/HMDB_sub.sdf.gz", package = "CompoundDb")
cmps <- compound_tbl_sdf(fl)
cmps
```

```
## Column synonyms contains a list
cmps$synonyms

## If we provide the optional argument collapse, multiple entries will be
## collapsed.
cmps <- compound_tbl_sdf(fl, collapse = "|")
cmps
cmps$synonyms
```

---

createCompDb *Create a CompDb database*

Description

CompDb databases can be created with the createCompDb() or the emptyCompDb() functions, the former creating and initializing (filling) the database with existing data, the latter creating an empty database that can be subsequently filled with insertCompound() or insertSpectra() calls.

emptyCompDb() requires only the file name of the database that should be created as input and returns a CompDb representing the empty database.

createCompDb() creates a SQLite-based CompDb object/database from a compound resource provided as a data.frame or tbl. Alternatively, the name(s) of the file(s) from which the annotation should be extracted can be provided. Supported are SDF files (such as those from the *Human Metabolome Database* HMDB) that can be read using the compound_tbl_sdf() or LipidBlast files (see compound_tbl_lipidblast().

An additional data.frame providing metadata information including the data source, date, version and organism is mandatory. By default, the function will define the name of the database based on the provided metadata, but it is also possible to define this manually with the dbFile parameter.

Optionally MS/MS (MS2) spectra for compounds can be also stored in the database. Currently only MS/MS spectra from HMDB are supported. These can be downloaded in XML format from HMDB (http://www.hmdb.ca), loaded with the msms_spectra_hmdb() or msms_spectra_mona() function and passed to the function with the msms_spectra argument. See msms_spectra_hmdb() or msms_spectra_mona() for information on the expected columns and format.

Required columns for the data.frame providing the compound information (parameter x) are:

- "compound_id": the ID of the compound. Can be an integer or character. Duplicated IDs are supported (for compatibility reasons), but not suggested. No missing values allowed.
- "name": the compound's name.
- "inchi": the InChI of the compound.
- "inchikey": the InChI key.
- "formula": the chemical formula.
- "exactmass": the compound's (exact) mass.
- "synonyms": additional synonyms/aliases for the compound. Should be either a single character or a list of values for each compound.

Any additional columns in the provided data.frame (such as e.g. "smiles" providing the compound's SMILES) are also supported and will be inserted into the database table.

See e.g. compound_tbl_sdf() or compound_tbl_lipidblast() for functions creating such compound tables.

The table containing the MS2 spectra data should have the following format and columns:

- "spectrum_id": an arbitrary ID for the spectrum. Has to be an integer.
- "compound_id": the ID of the compound to which the spectrum can be associated with. This has to be present in the data.frame defining the compounds.
- "polarity": the polarity (as an integer, 0 for negative, 1 for positive, NA for not set).
- "collision_energy": the collision energy.
- "predicted": whether the spectrum was predicted or measured.
- "splash": the SPLASH of the spectrum.
- "instrument_type": the instrument type.
- "instrument": the name of the instrument.
- "precursor_mz": the precursor m/z (as a numeric).
- "mz": the m/z values.
- "intensity": the intensity values.

Only for columns "spectrum_id", "compound_id", "mz" and "intensity" a value has to be provided in each row of the data.frame. The others are optional. Note that the data.frame can be either in the format as in the example below (i.e. each row being one spectrum and columns "mz" and "intensity" being of type list each element being the m/z or intensity values of one spectrum) or in a *full* form, in which each row represents one *peak* and all columns except "mz" and "intensity" containing redundant information of each spectrum (hence columns "mz" and "intensity" being of type numeric).

The metadata data.frame is supposed to have two columns named "name" and "value" providing the following minimal information as key-value pairs (see make_metadata for a utility function to create such a data.frame):

- "source": the source from which the data was retrieved (e.g. "HMDB").
- "url": the url from which the original data was retrieved.
- "source_version": the version from the original data source (e.g. "v4").
- "source_date": the date when the original data source was generated.
- "organism": the organism. Should be in the form "Hsapiens" or "Mmusculus".

createCompDbPackage creates an R data package with the data from a CompDb object.

make_metadata() helps generating a metadata data.frame in the correct format expected by the createCompDb function. The function returns a data.frame.

Usage

```
createCompDb(x, metadata, msms_spectra, path = ".", dbFile = character())

createCompDbPackage(
  x,
  version,
  maintainer,
  author,
  path = ".",
  license = "Artistic-2.0"
)

make_metadata(
  source = character(),
  url = character(),
  source_version = character(),
  source_date = character(),
  organism = NA_character_
)

emptyCompDb(dbFile = character())
```

Arguments

x	For `createCompDb()`: `data.frame` or `tbl` with the compound annotations or `character` with the file name(s) from which the compound annotations should be retrieved. See description for details. For `createCompDbPackage()`: `character(1)` with the file name of the `CompDb` SQLite file (created by `createCompDb`).
metadata	For `createCompDb()`: `data.frame` with metadata information. See description for details.
msms_spectra	For `createCompDb()`: `data.frame` with MS/MS spectrum data. See `msms_spectra_hmdb()` for the expected format and a function to import such data from spectrum xml files from HMDB.
path	`character(1)` with the path to the directory where the database file or package folder should be written. Defaults to the current directory.
dbFile	`character(1)` to optionally provide the name of the SQLite database file. If not provided (the default) the database name is defined using information from the provided `metadata`.
version	For `createCompDbPackage()`: `character(1)` with the version of the package (ideally in the format `"x.y.z"`).
maintainer	For `createCompDbPackage()`: `character(1)` with the name and email address of the package maintainer (in the form `"First Last <first.last@provider.com>"`.
author	For `createCompDbPackage()`: `character(1)` with the name of the package author.

license	For createCompDbPackage(): character(1) with the license of the package respectively the originating provider.
source	For make_metadata(): character(1) with the name of the resource that provided the compound annotation.
url	For make_metadata(): character(1) with the url to the original resource.
source_version	For make_metadata(): character(1) with the version of the original resource providing the annotation.
source_date	For make_metadata(): character(1) with the date of the resource's release.
organism	For make_metadata(): character(1) with the name of the organism. This should be in the format "Hsapiens" for human, "Mmusculus" for mouse etc. Leave to NA if not applicable.

Details

Metadata information is also used to create the file name for the database file. The name starts with "CompDb", followed by the organism, the data source and its version. A compound database file for HMDB version 4 with human metabolites will thus be named: "CompDb.Hsapiens.HMDB.v4".

A single CompDb database is created from multiple SDF files (e.g. for *PubChem*) if all the file names are provided with parameter x. Parallel processing is currently not enabled because SQLite does not support it yet natively.

Value

For createCompDb(): a character(1) with the database name (invisibly).

Author(s)

Johannes Rainer

See Also

compound_tbl_sdf() and compound_tbl_lipidblast() for functions to extract compound annotations from files in SDF format, or files from LipidBlast.

import_mona_sdf() to import both the compound and spectrum data from a SDF file from MoNa (Massbank of North America) in one call.

msms_spectra_hmdb() and msms_spectra_mona() for functions to import MS/MS spectrum data from xml files from HMDB or an SDF file from MoNa.

CompDb() for how to use a compound database.

Examples

```
## Read compounds for a HMDB subset
fl <- system.file("sdf/HMDB_sub.sdf.gz", package = "CompoundDb")
cmps <- compound_tbl_sdf(fl)

## Create a metadata data.frame for the compounds.
metad <- data.frame(name = c("source", "url", "source_version",
```

```
    "source_date", "organism"), value = c("HMDB", "http://www.hmdb.ca",
    "v4", "2017-08-27", "Hsapiens"))

## Alternatively use the make_metadata helper function
metad <- make_metadata(source = "HMDB", source_version = "v4",
    source_date = "2017-08", organism = "Hsapiens",
    url = "http://www.hmdb.ca")
## Create a SQLite database in the temporary folder
db_f <- createCompDb(cmps, metadata = metad, path = tempdir())

## The database can be loaded and accessed with a CompDb object
db <- CompDb(db_f)
db

## Create a database for HMDB that includes also MS/MS spectrum data
metad2 <- make_metadata(source = "HMDB_with_spectra", source_version = "v4",
    source_date = "2017-08", organism = "Hsapiens",
    url = "http://www.hmdb.ca")

## Import spectrum information from selected MS/MS xml files from HMDB
## that are provided in the package
xml_path <- system.file("xml", package = "CompoundDb")
spctra <- msms_spectra_hmdb(xml_path)

## Create a SQLite database in the temporary folder
db_f2 <- createCompDb(cmps, metadata = metad2, msms_spectra = spctra,
    path = tempdir())

## The database can be loaded and accessed with a CompDb object
db2 <- CompDb(db_f2)
db2

## Does the database contain MS/MS spectrum data?
hasMsMsSpectra(db2)

## Create a database for a ChEBI subset providing the file name of the
## corresponding SDF file
metad <- make_metadata(source = "ChEBI_sub", source_version = "2",
    source_date = NA, organism = "Hsapiens", url = "www.ebi.ac.uk/chebi")
db_f <- createCompDb(system.file("sdf/ChEBI_sub.sdf.gz",
    package = "CompoundDb"), metadata = metad, path = tempdir())
db <- CompDb(db_f)
db

compounds(db)

## connect to the database and query it's tables using RSQlite
library(RSQLite)
con <- dbConnect(dbDriver("SQLite"), db_f)

dbGetQuery(con, "select * from metadata")
dbGetQuery(con, "select * from ms_compound")
```

```
## To create a CompDb R-package we could simply use the
## createCompDbPackage function on the SQLite database file name.
```

`expandMzIntensity` *Expand m/z and intensity values in a data.frame*

Description

`expandMzIntensity()` *expands* a `data.frame` with m/z and/or intensity values stored as a `list` in columns `"mz"` and `"intensity"`. The resulting `data.frame` has the m/z and intensity values stored as `numeric` in columns `"mz"` and `"intensity"`, one value per row, with the content of other columns repeated as many times as there are m/z and intensity values.

Usage

```
expandMzIntensity(x)
```

Arguments

`x`	`data.frame` with *collapsed* m/z and intensity values in columns `"mz"` and `"intensity"`, such as returned by `msms_spectra_hmdb()` with parameter `collapsed = TRUE`, or by `spectra` or `compounds` calls.

Value

`data.frame` with `"mz"` and `"intensity"` columns *expanded*. See description for details.

Author(s)

Johannes Rainer

Examples

```
## Read a data.frame with collapsed columns mz and intensity columns
dr <- system.file("xml/", package = "CompoundDb")
msms_spctra <- msms_spectra_hmdb(dr)

msms_spctra

## Columns mz and intensity are "collased"
msms_spctra$mz

## Expand the data.frame to get one row per m/z and intensity value
spctra_exp <- expandMzIntensity(msms_spctra)
spctra_exp
```

`Filter-classes` *Filters supported by CompDb and IonDb*

Description

A variety of different filters can be applied to the `CompDb` object to retrieve only subsets of the data. These filters extend the AnnotationFilter::AnnotationFilter class and support the filtering concepts introduced by Bioconductor's `AnnotationFilter` package.

The supported filters are:

- `CompoundIdFilter`: filter based on the compound ID.
- `FormulaFilter`: filter based on the compound's formula.
- `InchiFilter`: filter based on the compound's InChI.
- `InchikeyFilter`: filter based on the compound's InChI key.
- `ExactmassFilter`: filter based on the compound's (exact) mass.
- `NameFilter`: filter based on the compound name.
- `MsmsMzRangeMinFilter`: retrieve entries based on the smallest m/z of all peaks of their MS/MS spectra. Requires that MS/MS spectra data are present (i.e. `hasMsMsSpectra(cmp_db)` returns `TRUE`).
- `MsmsMzRangeMaxFilter`: retrieve entries based on the largest m/z of all peaks of their MS/MS spectra. Requires that MS/MS spectra data are present (i.e. `hasMsMsSpectra(cmp_db)` returns `TRUE`).
- `SpectrumIdFilter`: retrieve entries associated with the provided IDs of MS/MS spectra.

In addition to the filters listed above, the following ones are supported by a IonDb (but not by a CompDb):

- `IonIdFilter`: filter based on the ion ID.
- `IonAdductFilter`: filter based on the adduct.
- `IonMzFilter`: filter based on the mz of the ion.
- `IonRtFilter`: filter based on the rt of the ion.

Usage

```
CompoundIdFilter(value, condition = "==")

SpectrumIdFilter(value, condition = "==")

NameFilter(value, condition = "==")

MsmsMzRangeMinFilter(value, condition = ">=")

MsmsMzRangeMaxFilter(value, condition = "<=")
```

```
ExactmassFilter(value, condition = "==")

FormulaFilter(value, condition = "==")

InchiFilter(value, condition = "==")

InchikeyFilter(value, condition = "==")

IonIdFilter(value, condition = "==")

IonAdductFilter(value, condition = "==")

IonMzFilter(value, condition = "==")

IonRtFilter(value, condition = "==")
```

Arguments

value	The value for the filter. For details see AnnotationFilter::AnnotationFilter().
condition	The condition for the filter. For details see AnnotationFilter::AnnotationFilter().

Value

Constructor functions return an instance of the respective class.

Author(s)

Johannes Rainer

See Also

AnnotationFilter::supportedFilters() for the method to list all supported filters for a CompDb (or a IonDb) object.

Examples

```
## Create a filter for the compound id
cf <- CompoundIdFilter("comp_a")
cf

## Create a filter using a formula expression
AnnotationFilter(~ compound_id == "comp_b")

## Combine filters
AnnotationFilterList(CompoundIdFilter("a"), NameFilter("b"))

## Using a formula expression
AnnotationFilter(~ compound_id == "a" | name != "b")
```

`import_mona_sdf` *Import compound and spectrum information from MoNa*

Description

`import_mona_sdf()` allows to import compound and spectrum information from an SDF file from MoNa (Massbank of North America http://mona.fiehnlab.ucdavis.edu/). This function is a convenience function using the `compound_tbl_sdf()` and `msms_spectra_mona()` functions for data import but avoiding to read the SDF files twice.

Usage

```
import_mona_sdf(x, nonStop = TRUE)
```

Arguments

`x`	`character(1)` being the SDF file name.
`nonStop`	`logical(1)` whether file content specific errors should only reported as warnings and not break the full import process. The value of this parameter is passed to parameter `skipErrors` of the `ChemmineR::read.SDFset()` function.

Value

A `list` with elements `"compound"` and `"msms_spectrum"` containing data.frames with compound and MS/MS spectrum data, respectively.

Note

MoNa SDF files organize the data by individual spectra (i.e. each element is one spectrum) and individual compounds can not easily and consistently defined (i.e. not all entries have an InChI ID or other means to uniquely identify compounds). Thus, the function returns a highly redundant compound table. Feedback on how to reduce this redundancy would be highly welcome!

Author(s)

Johannes Rainer

See Also

`compound_tbl_sdf()` to read only the compound information.

`msms_spectra_mona()` to read only the spectrum data.

Examples

```
## Define the test file containing a small subset from MoNa
fl <- system.file("sdf/MoNa_export-All_Spectra_sub.sdf.gz",
    package = "CompoundDb")

## Import the data
res <- import_mona_sdf(fl)
```

IonDb *IonDb: compound database with additional ion information*

Description

IonDb objects extends CompDb by allowing to store also information about measured ions to a CompDb() database. This information includes the type (adduct) of the ion, it's measured (or expected) retention time for a certain LC-MS setup and its mass-to-charge ratio.

As suggested use case, users might create (or download) a CompDb (SQLite) database e.g. containing compound (and eventually MS/MS spectra) annotations from public databases such as the Human Metabolome Database (HMDB) or MassBank. To store now measured ions (e.g. of lab-internal standards) for a certain LC-MS setup, such a CompDb can then be converted to an IonDb using the IonDb() constructor function. Ions can be subsequently added using the insertIon() function. In general, it is suggested to create one IonDb database for one specific LC-MS setup. Such an IonDb database can then be used to match experimental m/z and retention times against ions defined in the database (using the functionality of the MetaboAnnotation package).

Usage

```
## S4 method for signature 'IonDb'
ionVariables(object, includeId = FALSE, ...)

## S4 method for signature 'IonDb'
ions(
  object,
  columns = ionVariables(object),
  filter,
  return.type = c("data.frame", "tibble"),
  ...
)

## S4 method for signature 'IonDb'
insertIon(object, ions, addColumns = FALSE)

## S4 method for signature 'IonDb'
deleteIon(object, ids = integer(0), ...)

## S4 method for signature 'missing,missing'
IonDb(x, cdb, flags = SQLITE_RWC, ...)
```

```
## S4 method for signature 'CompDb,missing'
IonDb(x, cdb, ions = data.frame(), ...)

## S4 method for signature 'character,missing'
IonDb(x, cdb, flags = SQLITE_RW, ...)

## S4 method for signature 'DBIConnection,missing'
IonDb(
  x,
  cdb,
  ions = data.frame(),
  flags = SQLITE_RW,
  ...,
  .DBNAME = character()
)

## S4 method for signature 'character,CompDb'
IonDb(x, cdb, ions = data.frame(), flags = SQLITE_RW, ...)

## S4 method for signature 'DBIConnection,CompDb'
IonDb(
  x,
  cdb,
  ions = data.frame(),
  flags = SQLITE_RW,
  ...,
  .DBNAME = character()
)
```

Arguments

object	For all methods: a IonDb object.
includeId	For ionVariables(): logical(1) whether the ion ID (column "ion_id") should be included in the result. The default is includeId = FALSE.
...	additional arguments. Currently not used.
columns	For ions(): character with the names of the database columns that should be retrieved. Use ionVariables for a list of available column names.
filter	For ions(): filter expression or AnnotationFilter::AnnotationFilter() defining a filter to be used to retrieve specific elements from the database.
return.type	For ions(): either "data.frame" or "tibble" to return the result as a data.frame() or tibble::tibble(), respectively.
ions	for insertIon() and IonDb(): data.frame with ion definitions to be added to the IonDb database. Columns "compound_id" (character()), "ion_adduct" (character()), "ion_mz" (numeric()) and "ion_rt" (numeric()) are mandatory (but, with the exception of "compound_id", can contain NA).

addColumns
: For insertIons(): logical(1) whether columns being present in the submitted data.frame but not in the database table should be added to the database's ion table.

ids
: For deleteIon(): character() or (alternatively integer()) specifying the IDs of the ions to delete. IDs in ids that are not associated to any ion in the IonDb object are ignored.

x
: For IonDb(): database connection or character(1) with the file name of the SQLite database where the IonDb data will be stored or a CompDb() object that should be converted into an IonDb object.

  For all other methods: an `IonDb` object.

cdb
: For IonDb(): CompDb object from which data should be transferred to the IonDb database.

flags
: For IonDb(): optional integer(1) defining the flags for the SQLite database connection. Only used if x is a character().

.DBNAME
: character(1) defining the SQLite database file. This is an internal parameter not intended to be used/provided by the user.

Value

See description of the respective function.

Creation of IonDb objects/databases

- A new IonDb database can be created and initialized with data from an existing CompDb database by passing either the database connection (e.g. an SQLiteConnection) or the file path of a (to be created) SQLite database with parameter x to the IonDb() function and the CompDb object with parameter cdb. Optional parameter ions allows insert in addition ion definitions (which can also be added later using insertIon() function calls).
- An existing CompDb can be converted to an IonDb by passing the CompDb() object with parameter x to the IonDb function. Optional parameter ions allows to provide a data.frame with ion definitions to be inserted in to the database (which can also be added later using insertIon() function calls). Note that this fails if the database connection for the CompDb is read-only.
- Previously created IonDb databases can be loaded by passing either the database connection (e.g. an SQLiteConnection) or the file path of the (SQLite) database with parameter x to the IonDb() function.

Retrieve annotations and ion information from the database

Annotations/compound informations can be retrieved from a IonDb in the same way as thay are extracted from a CompDb. In addition, the function ions() allows to retrieve the specific ion information from the database. It returns the actual data as a data.frame (if return.type = "data.frame") or a tibble::tibble() (if return.type = "tibble"). An ions() call will always return all elements from the *ms_ion* table (unless a filter is used).

General functions (beside those inherited from CompDb)

- IonDb(): connect to or create a compound/ion database.
- ionVariables(): returns all available columns/database fields for ions.

Adding and removing data from a database

IonDb inherits the insertCompound(), insertSpectra(), deleteCompound() and deleteSpectra() functions from CompDb(). In addition, IonDb defines the functions:

- insertIon(): adds ions to the IonDb object. Note that insertIon() always adds all the ions specified through the ions parameter and does not check if they are already in the database. To add columns present in the submitted data.frame to the database table set addColumns = TRUE (default is addColumns = FALSE).
- deleteIon(): deletes ions from the IonDb object by specifying their IDs.

Filtering the database

Like compounds() and Spectra() also ions() allows to filter the results using specific filter classes and expressions. Filtering uses the concepts from Bioconductor's *AnnotationFilter* package. All information for a certain compound with the ID "1" can for example be retrieved by passing the filter expression filter = ~ ion_id == 1 to the ions() function.

Use the AnnotationFilter::supportedFilters() function on the IonDb object to get a list of all supported filters. See also examples below or the usage vignette for details.

Author(s)

Andrea Vicini, Johannes Rainer

Examples

```
# We load a small compound test database based on MassBank which is
# distributed with this package.
cdb <- CompDb(system.file("sql/CompDb.MassBank.sql", package = "CompoundDb"))
cdb

# We next want to convert this CompDb into an IonDb, but the original CompDb
# database is read only, thus we have to provide the name (or connection) of
# an other database to transfer all the data from the CompDb to that.
idb <- IonDb(paste0(tempdir(), "/idb_ex.db"), cdb)
idb

# It is also possible to load a previously created IonDb passing only the
# connection to the database.
idb2 <- IonDb(paste0(tempdir(), "/idb_ex.db"))

# Ion definitions can be added to the database with the `insertIon` function
# providing a `data.frame` with ion definition. This `data.frame` is expected
# to provide the IDs of the compounds, an adduct name/definition and the
# (experimentally determined) m/z and retention time of the ion. To list
# compound IDs from the CompDb database:
```

```
head(compounds(cdb, c("compound_id", "name")))

ions = data.frame(compound_id = c("1", "1", "2", "3", "6", "35"),
                  ion_adduct = c("[M+H]+", "[M+Na]+", "[M+Na]+",
                                 "[M+Na]+", "[M+2H]2+", "[M+H-NH3]+"),
                  ion_mz = c(179.0703, 201.0522, 201.0522,
                             201.0522, 253.66982, 312.0390),
                  ion_rt = 1:6)

# Inserting ion definitions.
idb <- insertIon(idb, ions)
idb

ions(idb, columns = c("name", "formula", "ion_adduct", "ion_mz", "ion_rt"))

## List all available ion variables
ionVariables(idb)

## Extract a data.frame with ion variables for all ions
ions(idb)

## List all database tables and their columns
tables(idb)

## Filtering the database
##
## Get all ions with an m/z between 200 and 300
res <- ions(idb, filter = ~ ion_mz > 200 & ion_mz < 300)
res

## Get all ions that have a H in their adduct definition.
res <- ions(idb, filter = IonAdductFilter("H", "contains"))
res
```

MsBackendCompDb *CompDb-based MS spectrum backend*

Description

The MsBackendCompDb *represents* MS2 spectra data from a CompDb() object/database. The object keeps only the primary keys of the spectra, the associated compound IDs and the precursor m/z values in memory and has thus only a very low memory footprint. All spectra variables, including m/z and intensity values are retrieved from the database *on-demand*. By extending the Spectra::MsBackendCached() class directly, MsBackendCompDb supports adding/replacing spectra variables. These values are however only cached within the object and not propagated (written) to the database.

It is not intended that users create or use instances of this class directly, the Spectra::Spectra() call on CompDb() will return a Spectra object that uses this backend.

The MsBackendCompDb does not support parallel processing because the database connection stored within the object can not be used across multiple parallel processes. The backendBpparam() method for MsBackendCompDb thus returns always SerialParam and hence any function that uses this method to check for parallel processing capability of a MsBackend will by default disable parallel processing.

Usage

```
MsBackendCompDb()

## S4 method for signature 'MsBackendCompDb'
backendInitialize(object, x, filter, ...)

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

## S4 method for signature 'MsBackendCompDb'
peaksData(object, columns = c("mz", "intensity"))

## S4 method for signature 'MsBackendCompDb'
peaksVariables(object)

## S4 method for signature 'MsBackendCompDb'
dataStorage(object)

## S4 replacement method for signature 'MsBackendCompDb'
intensity(object) <- value

## S4 replacement method for signature 'MsBackendCompDb'
mz(object) <- value

## S4 method for signature 'MsBackendCompDb'
spectraData(object, columns = spectraVariables(object))

## S4 method for signature 'MsBackendCompDb'
spectraNames(object)

## S4 replacement method for signature 'MsBackendCompDb'
spectraNames(object) <- value

## S4 method for signature 'MsBackendCompDb,ANY'
x[i, j, ..., drop = FALSE]

## S4 method for signature 'MsBackendCompDb,ANY'
extractByIndex(object, i)

## S4 replacement method for signature 'MsBackendCompDb'
x$name <- value
```

```
## S4 method for signature 'MsBackendCompDb'
tic(object, initial = TRUE)

## S4 method for signature 'MsBackendCompDb'
backendBpparam(object, BPPARAM = bpparam())
```

Arguments

object	an MsBackendCompDb instance.
x	an MsBackendCompDb instance.
filter	for backendInitialize(): optional filter expression to specify which elements to retrieve from the database.
...	ignored.
columns	for spectraData(): character with names of columns/spectra variables that should be returned. Defaults to spectraVariables(object). Database columns "ms_level", "precursor_mz", "precursor_intensity", "precursor_charge" are mapped to the core Spectra variables msLevel, precursorMz, precursorIntensity and precursorCharge, respectively. For peaksData: character with the names of the peaks columns to return. Use peaksVariables for supported values.
value	for $<-: the replacement values.
i	For [: integer, logical or character to subset the object.
j	For [: not supported.
drop	For [: not considered.
name	for $<-: the name of the spectra variable to replace.
initial	for tic(): logical(1) whether original total ion current values should be returned or if the values should be calculated based on the actual intensity values of each spectrum.
BPPARAM	for backendBpparam(): BiocParallel parallel processing setup. See BiocParallel::bpparam() for more information.

Value

See the description of the respective function.

Methods implemented for MsBackendCompDb

The methods listed here are implemented for the MsBackendCompDb. All other methods are inherited directly from the parent Spectra::MsBackendCached() class. See the help of Spectra::MsBackend() in the Spectra package for a complete listing of methods.

- peaksData(): gets the full list of peak matrices. Returns a list(), length equal to the number of spectra and each element being a matrix with columns "mz" and "intensity" with the spectra's m/z and intensity values.
- peaksVariables(): lists the available peaks variables in the backend (database). These can be used for parameter columns of peaksData().
- intensity<-: not supported.

- `mz<-`: not supported.
- `spectraData()`: returns the complete spectrum data including m/z and intensity values as a `S4Vectors::DataFrame()`.
- `$<-`: replace or add a spectrum variable. Note that `mz`, `intensity` and `spectrum_id` variables can not be replaced.
- `spectraNames()`: returns values from *spectrum_id* database column.

Note

For higher performance it is suggested to change the backend of the `Spectra::Spectra()` object to an `Spectra::MsBackendMemory()` backend with the `Spectra::setBackend()` method of `Spectra` objects.

Author(s)

Johannes Rainer

Examples

```
## MsBackendCompDb are not expected to be created/instanciated by users
## directly. Users also almost never directly interact with this type of
## object, as it is intented as a pure data backend for the `Spectra` object.
## Users will thus access MS data through such `Spectra` object, which can
## be created for `CompDb` objects using the `Spectra` method (see help
## of the `CompDb` object for more information. This examples shows how
## a `MsBackendCompDb` could be created purely from an SQLite database
## with data from a CompoundDb database.

## Connect to the SQLite database of a `CompDb` distributed via this package
library(RSQLite)
library(Spectra)
cdb <- CompDb(system.file("sql/CompDb.MassBank.sql", package = "CompoundDb"))

be <- backendInitialize(MsBackendCompDb(), cdb)
be

## Accessing m/z values
mz(be)
```

---

msms_spectra_hmdb *Import MS/MS spectra from HMDB xml files*

Description

`msms_spectra_hmdb()` imports MS/MS spectra from corresponding xml files from HMDB (http://www.hmdb.ca) and returns the data as a `data.frame`. HMDB stores MS/MS spectrum data in xml files, one file per spectrum.

Depending on the parameter `collapsed`, the returned `data.frame` is either *collapsed*, meaning that each row represents data from one spectrum xml file, or *expanded* with one row for each m/z and intensity pair for each spectrum. Columns `"mz"` and `"intensity"` are of type `list` for `collapsed = TRUE` and `numeric` for `collapsed = FALSE`.

Usage

```
msms_spectra_hmdb(x, collapsed = TRUE)
```

Arguments

`x`	`character(1)`: with the path to directory containing the xml files.
`collapsed`	`logical(1)` whether the returned `data.frame` should be *collapsed* or *expanded*. See description for more details.

Value

`data.frame` with as many rows as there are peaks and columns:

- spectrum_id (`integer`): an arbitrary, unique ID identifying values from one xml file.
- original_spectrum_id (`character`): the HMDB-internal ID of the spectrum.
- compound_id (`character`): the HMDB compound ID the spectrum is associated with.
- polarity (`integer`): 0 for negative, 1 for positive, `NA` for not set.
- collision_energy (`numeric`): collision energy voltage.
- predicted (`logical`): whether the spectrum is predicted or experimentally verified.
- splash (`character`): the SPLASH (SPectraL hASH) key of the spectrum (Wohlgemuth 2016).
- instrument_type (`character`): the type of MS instrument on which the spectrum was measured.
- instrument (`character`): the MS instrument (not available for all spectra in HMDB).
- precursor_mz (`numeric`): not provided by HMDB and thus `NA`.
- mz (`numeric` or `list` of `numeric`): m/z values of the spectrum.
- intensity (`numeric` or `list` of `numeric`): intensity of the spectrum.

Note

The HMDB xml files are supposed to be extracted from the downloaded zip file into a folder and should not be renamed. The function identifies xml files containing MS/MS spectra by their file name.

The same spectrum ID can be associated with multiple compounds. Thus, the function assignes an arbitrary ID (column `"spectrum_id"`) to values from each file. The original ID of the spectrum in HMDB is provided in column `"original_spectrum_id"`.

Author(s)

Johannes Rainer

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, Selak RM, Steinbeck C, Nainala VC, Mistrik R, Nishioka T, Fiehn O. SPLASH, A hashed identifier for mass spectra. Nature Biotechnology 2016 34(11):1099-1101

See Also

`createCompDb()` for the function to create a CompDb database with compound annotation and spectrum data.

Other spectrum data import functions.: `msms_spectra_mona()`

Examples

```
## Locate the folder within the package containing test xml files.
pth <- system.file("xml", package = "CompoundDb")

## List all files in that directory
dir(pth)

## Import spectrum data from HMDB MS/MS spectrum xml files in that directory
msms_spectra_hmdb(pth)

## Import the data as an *expanded* data frame, i.e. with a row for each
## single m/z (intensity) value.
msms_spectra_hmdb(pth, collapsed = FALSE)
```

---

msms_spectra_mona *Import MS/MS spectra from MoNa*

Description

`msms_spectra_mona()` imports MS/MS spectra from a MoNa (Massbank of North America, http://mona.fiehnlab.ucdavis.edu
SDF file and returns the data as a `data.frame`.

Depending on the parameter `collapsed`, the returned `data.frame` is either *collapsed*, meaning that each row represents data from one spectrum, or *expanded* with one row for each m/z and intensity pair for each spectrum. Columns `"mz"` and `"intensity"` are of type `list` for `collapsed = TRUE` and `numeric` for `collapsed = FALSE`.

Usage

```
msms_spectra_mona(x, collapsed = TRUE)
```

Arguments

`x`	`character(1)`: with the path to directory containing the xml files.
`collapsed`	`logical(1)` whether the returned `data.frame` should be *collapsed* or *expanded*. See description for more details.

Value

`data.frame` with as many rows as there are peaks and columns:

- spectrum_id (`integer`): an arbitrary, unique ID for each spectrum.
- original_spectrum_id (`character`): The ID from the spectrum as specified in the MoNa SDF.
- compound_id (`character`): the compound ID the spectrum is associated with.
- polarity (`integer`): 0 for negative, 1 for positive, `NA` for not set.
- collision_energy (`character`): collision energy voltage.
- predicted (`logical`): whether the spectrum is predicted or experimentally verified.
- splash (`character`): `NA` since SPLASH (SPectraL hASH) keys are not provided.
- instrument_type (`character`): the type of MS instrument on which the spectrum was measured.
- instrument (`character`): the MS instrument.
- precursor_mz (`numeric`): precursor m/z.
- adduct (`character`): ion formed from the precursor ion.
- ms_level (`integer`): stage of the sequential mass spectrometry (MSn).
- mz (`numeric` or `list` of `numeric`): m/z values of the spectrum.
- intensity (`numeric` or `list` of `numeric`): intensity of the spectrum.

Note

The identifiers provided by MoNa are used as *original_spectrum_id*. Note also that the MoNa data is not normalized in the sense that each spectrum is associated to one compound and the compound data is partially redundant. Also, MoNa does not provide a *splash* for a spectrum, hence the corresponding column will only contain `NA`.

Author(s)

Johannes Rainer

See Also

`createCompDb()` for the function to create a CompDb database with compound annotation and spectrum data.

Other spectrum data import functions.: `msms_spectra_hmdb()`

Examples

```
## Define the test file containing the data
fl <- system.file("sdf/MoNa_export-All_Spectra_sub.sdf.gz",
    package = "CompoundDb")
## Import spectrum data from the SDF file with a subset of the MoNa data
msms_spectra_mona(fl)

## Import the data as an *expanded* data frame, i.e. with a row for each
## single m/z (intensity) value.
msms_spectra_mona(fl, collapsed = FALSE)
```

Index