Package: MsIO (via r-universe)

September 5, 2024

Title Serializing and restoring/importing mass spectrometry data objects

Version 0.0.4

Description The serialization mechanism of R allows to save and load R data objects in a binary format, that can however not be read by other programming languages or software. The MsIO package supports serializing and restoring or importing mass spectrometry data objects to and from language agnostic file formats. A variety of different file types, including HDF5 and JSON-based formats defined by the Bioconductor *alabaster* package are supported. The file type can be defined and configured through a second argument `param` of the export/import methods.

Depends R (>= 4.2.0)

Imports jsonlite, methods, MsCoreUtils (>= 1.17.1), S4Vectors, ProtGenerics, alabaster.base

Suggests BiocStyle (>= 2.5.19), faahKO, knitr (>= 1.1.0), MsExperiment, msdata, rmarkdown, roxygen2, Spectra (>= 1.15.6), testthat, xcms, alabaster.se, alabaster.matrix

License Artistic-2.0

Encoding UTF-8

VignetteBuilder knitr

BugReports `https://github.com/RforMassSpectrometry/MsIO/issues`

URL `https://github.com/RforMassSpectrometry/MsIO`

biocViews Infrastructure, MassSpectrometry, Metabolomics, DataImport, Proteomics

Roxygen list(markdown=TRUE)

RoxygenNote 7.3.2

Collate 'PlainTextParam.R' 'AlabasterParam.R' 'AllGenerics.R' 'MetaboLightsParam.R' 'MsBackend.R' 'MsBackendMzR.R' 'MsExperiment.R' 'MsExperimentFiles.R' 'Spectra.R' 'mzTabParam.R' 'XcmsExperiment.R' 'zzz.R'

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

RemoteUrl https://github.com/rformassspectrometry/MsIO

RemoteRef HEAD

RemoteSha 1bddd738788bb84baee927d123b41cffe6b825da

Contents

`AlabasterParam` *Store MS data objects using the alabaster framework*

Description

The *alabaster* framework provides the methodology to save R objects to on-disk representations/ storage modes which are programming language independent (in contrast to e.g. R's RDS files). By using standard file formats such as JSON and HDF5, alabaster ensures that the data can also be read and imported by other programming languages such as Python or Javascript. This improves interoperability between application ecosystems.

The *alabaster* package defines the `saveObject()` and `readObject()` methods. Implementations of these methods are available for the following classes hence allowing to use `saveObject()` and `readObject()` directly on these objects:

- `MsBackendMzR`, defined in the *Spectra* package.
- `Spectra`, defined in the *Spectra* package.

In addition, the *MsIO* package defines the `AlabasterParam` which can be used to write or read MS objects using the `saveMsObject()` and `readMsObject()` methods. This allows additional configurations and customizations to the export or import process. It is thus for example possible to specify the path to the original MS data files for *on-disk* MS representations such as the `MsBackendMzR` which enables to import a stored object even if either the object or the original MS data files have been moved to a different directory or file system.

Importantly, it is only possible to save **one object in one directory**. To overwrite an existing stored object in a folder, that folder has to be deleted beforehand.

Details and properties for the *alabaster*-based storage modes for the various supported MS data objects are listed in the following sections.

Usage

```
AlabasterParam(path = tempdir())

## S4 method for signature 'MsBackendMzR'
saveObject(x, path, ...)

## S4 method for signature 'MsBackendMzR,AlabasterParam'
saveMsObject(object, param)

## S4 method for signature 'MsBackendMzR,AlabasterParam'
readMsObject(object, param, spectraPath = character())

## S4 method for signature 'MsExperiment'
saveObject(x, path, ...)

## S4 method for signature 'MsExperiment,AlabasterParam'
saveMsObject(object, param)

## S4 method for signature 'MsExperiment,AlabasterParam'
readMsObject(object, param, ...)

## S4 method for signature 'Spectra'
saveObject(x, path, ...)

## S4 method for signature 'Spectra,AlabasterParam'
saveMsObject(object, param)

## S4 method for signature 'Spectra,AlabasterParam'
readMsObject(object, param, ...)

## S4 method for signature 'XcmsExperiment'
saveObject(x, path, ...)

## S4 method for signature 'XcmsExperiment,AlabasterParam'
saveMsObject(object, param)

## S4 method for signature 'XcmsExperiment,AlabasterParam'
readMsObject(object, param, ...)
```

Arguments

`path`	`character(1)` with the name of the directory where the MS data object should be saved to or from which it should be restored. Importantly, path should point to a **new** folder, i.e. a directory that **does not already exist**.
`x`	MS data object to export. Can be one of the supported classes listed below.
`...`	optional additional parameters passed to the downstream functions, such as for example `spectraPath` described above.

`object`	for `saveMsObject()`: the MS data object to save, for `readMsObject()`: the MS data object that should be returned
`param`	an object defining and (eventually configuring) the file format and file name or directory to/from which the data object should be exported/imported.
`spectraPath`	For `readMsObject()`: `character(1)` optionally allowing to define the (absolute) path where the spectra files (*data storage files*) can be found. This parameter is used for `MsBackendMzR` (see descriptions below) and can be passed through `...` also to `readMsObject()` functions for other classes (such as `Spectra`, `MsExperiment` etc).

Value

For `AlabasterParam()`: an instance of `AlabasterParam` class. For `readObject()` the exported object in the specified path (depending on the type of object defined in the *OBJECT* file in the path. For `readMsObject()` the exported data object, defined with the function's first parameter, from the specified path. `saveObject()` and `saveMsObject()` don't return anything.

On-disk storage for `MsBackendMzR` objects

`MsBackendMzR` objects can be exported or imported using the `saveMsObject()` or `readMsObject()` functions to and from *alabaster*-based storage modes using the `AlabasterParam` parameter object. Alternatively *alabaster*'s `saveObject()` and `readObject()` can be used. The parameter `spectraPath` allows to define an alternative path to the original data files (in case these were moved). This parameter can be passed as additional parameter to both the `readObject()` as well as the `readMsObject()` methods.

The format of the folder contents follows the *alabaster* format: a file *OBJECT* (in JSON format) defines the type of object that was stored in the directory while the object's data, for `MsBackendMzR`, is stored in sub-folders *peaks_variables* (a `character` with the names of the peaks variables of the object) and *spectra_data* (the metadata for all spectra). Each sub-folder contains also an *OBJECT* file defining the object's type and an additional file (in HDF5 format) containing the data. See examples below for details.

On-disk storage for `Spectra` objects

`Spectra` objects can be exported/imported using `saveMsObject()` and `readMsObject()` with an `AlabasterParam`, or using the `saveObject()` and `readObject()` functions. Both read functions allow to pass additional parameters (such as `spectraPath`) to the import function for the `Spectra`'s backend.

The content of the folder with the stored `Spectra` data contains the *OBJECT* file defining the type of the object stored in that directory and the *spectra_processing_queue.json* file that contains the *processing queue* of the `Spectra` objects. All other slots of the object are saved in *alabaster* format into their respective sub-directories: *backend* for the `MsBackend` (see also `MsBackendMzR` above), *metadata* for the metadata slot, *processing* for the processing log, *processing_chunk_size* with the size for chunk-wise processing and *processing_queue_variables* for spectra/peaks variables that are needed for the processing queue.

On-disk storage for `MsExperiment` objects

`MsExperiment` is a container for various (different) MS data objects related to the same *experiment*. It is a very flexible object that can, but does not must contain actual MS data in form of e.g. a `Spectra` object. For the alabaster-based disk storage of an `MsExperiment`, each of the object's slots gets exported separately into its own subfolder within the object's directory (defined with parameter `path`). For the export of the individual slots, the respective `saveObject()` method is used. Similar to all other objects listed here, `MsExperiment` can be stored using either `saveObject()` or `saveMsObject` (with `AlabasterParam`) and *restored* using `readObject()` or `readMsObject()` (with `MsExperiment()` passed as the first parameter and `AlabasterParam` as second). The read functions support passing additional parameters to the import function(s) for object's MS data object(s), such as the `spectraPath` parameter described above through `...`.

The content of the folder with the stored `MsExperiment` data contains a file `OBJECT` (in JSON format, with the type of class defined as `"ms_experiment"`) and subfolders for the various slots, each saved to disk using the data type-specific `saveObject()` function:

- `@sampleData`: `DataFrame` stored into a folder named *sample_data*.
- `@sampleDataLinks`: the `List` is stored into a folder named *sample_data_links*, its *metadata columns* `DataFrame` (i.e. `mcols()` of the `List`) into a folder named *sample_data_links_mcols*.
- `@spectra`: if not `NULL, a Spectraobject stored into a folder with the name *spectra* (using`saveObject()`ofS` no directory *spectra* is created.
- `@experimentFiles`: `MsExperimentFiles` object saved using `saveObject()` into a folder named *experiment_files*. `MsExperimentFiles` are saved as a named list of `character` strings.
- `@qdata`: if not `NULL`, the object in this slot (either a `QFeatures` or `SummarizedExperiment`) is stored into a folder with the name *qdata* using the `saveObject()` method of the respective object. If the value for the `@qdata` slot is `NULL` the folder *qdata* is not created. At present, export of `QFeatures` objects is not supported!
- `@otherData`: `List` data is saved into a folder named *other_data*.
- `@metadata`: `List` data is saved into a filder named *metadata*.

Note that the data type of the `assays` of imported (previously stored) `SummarizedExperiment` objects are of type `ReloadedMatrix`.

On-disk storage for `XcmsExperiment` objects

`XcmsExperiment` objects extend the `MsExperiment` object and contain in addition the results of a preprocessing of the MS data using the *xcms* package. These objects can be exported/imported in the formats used for *alabaster*-based storage using the `saveObject()` and `readObject()` functions as well as using `saveMsObject()` and `readMsObject()` with an `AlabasterParam` parameter object. As with all other methods, additional parameters can be passed with the `...` parameter (such as the `spectraData` parameter for import of a `MsBackendMzR` discussed above). The storage directory contains all files and folders created by the export of the `MsExperiment` (see above) and in addition the specific results of *xcms* from the respective slots of the object:

- `@chromPeaks`: this numeric matrix is stored in a folder names *chrom_peaks*.
- `@chromPeakData`: this `data.frame` is first converted to a `DataFrame` and then stored to a folder *chrom_peak_data* (in the *alabaster* format for `DataFrame`).
- `@featureDefinitions`: this `data.frame` is first converted to a `DataFrame` and then stored to a folder *feature_definitions* (also in *alabaster* format for `DataFrame`).

- `@processHistory`: the list of `ProcessHistory` objects is stored in JSON format to a file *xcms_experiment_process_history.json*.

Author(s)

Johannes Rainer, Philippine Louail

See Also

Other MS object export and import formats.: `PlainTextParam`, `mzTabParam`

Examples

```
########
## Export and import a `MsBackendMzR` object:
####

library(Spectra)
library(msdata)
fl <- system.file("TripleTOF-SWATH", "PestMix1_DDA.mzML", package = "msdata")
be <- backendInitialize(MsBackendMzR(), fl)
be

## Export the object to a temporary directory using the alabaster framework;
## the equivalent command using the parameter object would be
## `saveMsObject(be, AlabasterParam(d))`.
d <- file.path(tempdir(), "ms_backend_mzr_example")
saveObject(be, d)

## List the content of the folder
dir(d, recursive = TRUE)

## The data can be imported again using alabaster's readObject() function
be_in <- readObject(d)
be_in

## Alternatively, the data could be restored also using
be_in <- readMsObject(MsBackendMzR(), AlabasterParam(d))

all.equal(mz(be), mz(be_in))


########
## Export and import of `Spectra` objects:
####

## Create a `Spectra` object with a `MsBackendMzR` backend.
s <- Spectra(fl)

## Define the folder to which to export and export the object
d <- file.path(tempdir(), "spectra_example")
saveMsObject(s, AlabasterParam(d))
```

```
## List the content of the directory
dir(d, recursive = TRUE)

## Restore the `Spectra` object again
s_in <- readMsObject(Spectra(), AlabasterParam(d))
s_in

## Alternatively, it would also be possible to just import the
## `MsBackendMzR` of the `Spectra`:
be_in <- readMsObject(MsBackendMzR(), AlabasterParam(file.path(d, "backend")))
be_in


########
## Export and import of `MsExperiment` objects:
####

library(MsExperiment)

## Create a new `MsExperiment` with sample data and our previously defined
## `Spectra` as its MS data
m <- MsExperiment(
    sampleData = data.frame(name = c("a", "b"), index = 1:2),
    spectra = s)
m

d <- file.path(tempdir(), "ms_experiment_example")
saveObject(m, d)

## List directory content
dir(d)

## Restore the stored object
m_in <- readObject(d)

m_in


########
## Export and import of `XcmsExperiment` objects:
####

## `XcmsExperiment` objects extend `MsExperiment` to represent all
## data of an MS experiment and contain in addition the results
## of the preprocessing of the data with the *xcms* package. Below
## we load the *xcms* package and load an example result object from that
## package.
library(xcms)
x <- loadXcmsData()
x

## Store this result object to a folder
```

```
d <- file.path(tempdir(), "xcms_experiment_example")
saveMsObject(x, AlabasterParam(d))

dir(d)

## Restore the data; eventually needed additional parameters, such as
## `spectraPath` to restore a `MsBackendMzR` if the original data files
## have been moved, could be passed with the `...` parameter of
## `readMsExperiment()`.
x_in <- readMsObject(XcmsExperiment(), AlabasterParam(d))
x_in
```

MetaboLightsParam *Load content from a MetaboLights study*

Description

The `MetaboLightsParam` class and the associated `readMsObject()` method allow users to load an `MsExperiment` object from a study in the MetaboLights database (https://www.ebi.ac.uk/metabolights/index) by providing its unique study `studyId`. This function is particularly useful for importing metabolomics data into an `MsExperiment` object for further analysis within the R environment. It's important to note that this method can *only* be used for import into an R environenement using `readMsObject()`. It cannot be used with the `saveMsObject()` method.

If the study contains multiple assays, the user will be prompted to select which assay to load. The resulting `MsExperiment` object will include a `sampleData` slot populated with data extracted from the selected assay. Columns in the `sampleData` that contain only `NA` values are automatically removed, and an additional column is added to track the injection index.

Usage

```
MetaboLightsParam(studyId = character(1))

## S4 method for signature 'MsExperiment,MetaboLightsParam'
readMsObject(object, param, ...)
```

Arguments

`studyId`	`character(1)` The MetaboLights study studyId, which should start with "MTBL". This identifier uniquely specifies the study within the MetaboLights database.
`object`	for `saveMsObject()`: the MS data object to save, for `readMsObject()`: the MS data object that should be returned
`param`	an object defining and (eventually configuring) the file format and file name or directory to/from which the data object should be exported/imported.
`...`	additional optional arguments. See documentation of respective method for more information.

Value

(for now ?) A `MsExperiment` object with only the sampleData slots filled (will be updated when MetaboLightsBackend available ?).

Author(s)

Philippine Louail

See Also

- `MsExperiment` object, defined in the (MsExperiment) package.
- MetaboLights for accessing the MetaboLights database.

Examples

```
library(MsExperiment)
# Load a study with the studyId "MTBLS10035"
param <- MetaboLightsParam(studyId = "MTBLS10035")
ms_experiment <- readMsObject(MsExperiment(), param)
```

`mzTabParam` *Store xcms preprocessing results to a file in mzTab-M format.*

Description

The `saveMsObject()` and `readMsObject()` methods with the `mzTabParam` option enable users to save/load `XcmsExperiment` objects in Mz-Tab-m file format. Mainly the metadata (MTD) and Small molecule feature (SMF) tables will represent the `XcmsExperiment`. More specifically, `sampleData()` of the object will be stored in the metadata section (MTD) along with the user-inputed `studyId` and `polarity`. The `featureDefinitions()` will be stored in the small molecule feature (SMF) section but by default only the `mzmed`, `rtmed`, `rtmin` and `rtmax` are exported. More info avaialble in `featureDefinitions()` can be exported by specifying the `optionalFeatureColumns` parameter. The `featureValues()` will also be stored in the small molecule feature (SMF) section.

The small molecule summary section (SML) will be filled with `null` values as no annotation and identification of compound is performed in `xcms`.

Writing data to a folder that contains already exported data will result in an error.

Usage

```
mzTabParam(
  studyId = character(),
  polarity = c("positive", "negative"),
  sampleDataColumn = character(),
  path = tempdir(),
  optionalFeatureColumns = character(),
  ...
)

## S4 method for signature 'XcmsExperiment,mzTabParam'
saveMsObject(object, param)
```

Arguments

`studyId`	`character(1)` Will be both the `filename` of the object saved in mzTab-M format and the `mzTab-ID` in the file.
`polarity`	`character(1)` Describes the polarity of the experiment. Two inputs are possible, "positive" (default) or "negative".
`sampleDataColumn`	`character` strings corresponding to the column name(s) of the `sampleData()` of the `XcmsExperiment` object with the different *variables* of the experiment, for example it could be *"phenotype"*, *"sample_type"*, etc...
`path`	`character(1)` Define where the file is going to be stored. The default will be `tempdir()`.
`optionalFeatureColumns`	Optional columns from `featureDefinitions()` that should be exported too. For example it could be *"ms_level"*, *"npeaks"*, etc...
`...`	additional optional arguments. See documentation of respective method for more information.
`object`	for `saveMsObject()`: the MS data object to save, for `readMsObject()`: the MS data object that should be returned
`param`	an object defining and (eventually configuring) the file format and file name or directory to/from which the data object should be exported/imported.

Slots

`dots` Correspond to any optional parameters to be passed to the `featureValues()` function. (e.g. parameters `method` or `value`).

Note

This function was build so that the output fit the recommendation of mzTab-M file format version 2.0. These can be found here: (http://hupo-psi.github.io/mzTab/2_0-metabolomics-release/mzTab_format_specification_2_0-M_release.html)

Author(s)

Philippine Louail, Johannes Rainer

References

Hoffmann N, Rein J, Sachsenberg T, Hartler J, Haug K, Mayer G, Alka O, Dayalan S, Pearce JTM, Rocca-Serra P, Qi D, Eisenacher M, Perez-Riverol Y, Vizcaino JA, Salek RM, Neumann S, Jones AR. mzTab-M: A Data Standard for Sharing Quantitative Results in Mass Spectrometry

Metabolomics. Anal Chem. 2019 Mar 5;91(5):3302-3310. doi: 10.1021/acs.analchem.8b04310. Epub 2019 Feb 13. PMID: 30688441; PMCID: PMC6660005.

See Also

Other MS object export and import formats.: `AlabasterParam`, `PlainTextParam`

Examples

```
## Load a test data set with detected peaks, of class `XcmsExperiment`
library(xcms)
test_xcms <- loadXcmsData()

## Define param
param <- mzTabParam(studyId = "test",
                    polarity = "positive",
                    sampleDataColumn = "sample_type")

## Save as a mzTab-M file
saveMsObject(test_xcms, param)
```

`PlainTextParam` *Store contents of MS objects as plain text files*

Description

The `saveMsObject()` and `readMsObject()` methods with the `PlainTextParam` option enable users to save/load different type of mass spectrometry (MS) object as a collections of plain text files in/from a specified folder. This folder, defined with the `path` parameter, will be created by the `storeResults()` function. Writing data to a folder that contains already exported data will result in an error.

All data is exported to plain text files, where possible as tabulator delimited text files. Data is exported using R's `write.table()` function, thus, the text files will also contain row names (first column) as well as column names (header). Strings in the text files are quoted. Some information, in particular the content of *parameter* classes within the objects, is stored in JSON format instead.

The MS object currently supported for import and export with this parameter are:

- `MsBackendMzR` object, defined in the (Spectra) package.
- `Spectra` object, defined in the (Spectra) package.
- `MsExperiment` object, defined in the (MsExperiment) package.
- `XcmsExperiment` object, defined in the (xcms) package.

See their respective section below for details and formats of the exported files.

Usage

```
PlainTextParam(path = tempdir())

## S4 method for signature 'MsBackendMzR,PlainTextParam'
saveMsObject(object, param)

## S4 method for signature 'MsBackendMzR,PlainTextParam'
readMsObject(object, param, spectraPath = character())

## S4 method for signature 'MsExperiment,PlainTextParam'
saveMsObject(object, param)

## S4 method for signature 'MsExperiment,PlainTextParam'
readMsObject(object, param, ...)

## S4 method for signature 'Spectra,PlainTextParam'
saveMsObject(object, param)

## S4 method for signature 'Spectra,PlainTextParam'
readMsObject(object, param, ...)

## S4 method for signature 'XcmsExperiment,PlainTextParam'
saveMsObject(object, param)

## S4 method for signature 'XcmsExperiment,PlainTextParam'
readMsObject(object, param, ...)
```

Arguments

`path`	For `PlainTextParam()`: `character(1)`, defining where the files are going to be stored/ should be loaded from. The default is `path = tempdir()`.
`object`	for `saveMsObject()`: the MS data object to save, for `readMsObject()`: the MS data object that should be returned
`param`	an object defining and (eventually configuring) the file format and file name or directory to/from which the data object should be exported/imported.
`spectraPath`	For `readMsObject()`: `character(1)` optionally allowing to define the (absolute) path where the spectra files (*data storage files*) can be found. This parameter is used for `MsBackendMzR` (see descriptions below) and can be passed through `...` also to `readMsObject()` functions for other classes (such as `Spectra`, `MsExperiment` etc).
`...`	Additional parameters passed down to internal functions. E.g. parameter `spectraPath` (see above).

Value

For `PlainTextParam()`: a `PlainTextParam` class. `saveMsObject()` does not return anything but saves the object to collections of different plain text files to a folder. The `readMsObject()` method returns the restored data as an instance of the class specified with parameter `object`.

On-disk storage for `MsBackendMzR` objects

For `MsBackendMzR` objects, defined in the `Spectra` package, the following file is stored:

- The backend's `spectraData()` is stored in a tabular format in a text file named *ms_backend_data.txt*. Each row of this tab-delimited text file corresponds to a spectrum with its respective metadata in the columns.

On-disk storage for `Spectra` objects

For `Spectra` objects, defined in the `Spectra` package, the files listed below are stored. Any parameter passed to the `saveMsObject()` method using its `...` parameter are passed to the `saveMsObject()` call of the `Spectra`'s backend.

- The `processingQueueVariables`, `processing`, `processingChunkSize()`, and `backend` class information of the object are stored in a text file named *spectra_slots.txt*. Each of these slots is stored such that the name of the slot is written, followed by "=" and the content of the slot.
- The processing queue of the `Spectra` object, ensuring that any spectra data modifications are retained, is stored in a `json` file named *spectra_processing_queue.json*. The file is written such that each processing step is separated by a line and includes all information about the parameters and functions used for the step.
- The `Spectra`'s MS data (i.e. it's backend) is stored/exported using the `saveMsObject()` method of the respective backend type. Currently only backends for which the `saveMsObject()` method is implemented (see above) are supported.

On-disk storage for `MsExperiment` objects

For `MsExperiment` objects, defined in the `MsExperiment` package, the exported data and related text files are listed below. Any parameter passed to the `saveMsObject()` through `...` are passed to the `saveMsObject()` calls of the individual MS data object(s) within the `MsExperiment`.

Note that at present `saveMsObject()` with `PlainTextParam` does **not** export the full content of the `MsExperiment`, i.e. slots `@experimentFiles`, `@qdata`, `@otherData` and `@metadata` are currently not saved.

- The `sampleData()` is stored as a text file named *ms_experiment_sample_data.txt*. Each row of this file corresponds to a sample with its respective metadata in the columns.
- The links between the sample data and any other data within the `MsExperiment` are stored in text files named *ms_experiment_sample_data_links_....txt*, with "..." referring to the data slot to which samples are linked. Each file contains the mapping between the sample data and the elements in a specific data slot (e.g., `Spectra`). The files are tabulator delimited text files with two columns of integer values, the first representing the index of a sample in the objects `sampleData()`, the second the index of the assigned element in the respective object slot. The table "ms_experiment_element_metadata.txt" contains the metadata of each of the available mappings.
- If the `MsExperiment` contains a `Spectra` object with MS data, it's content is exported to the same folder using a `saveMsObject()` call on it (see above for details of exporting `Spectra` objects to text files).

On-disk storage for `XcmsExperiment` objects

For `XcmsExperiment` objects, defined in the *xcms* package, the exported data and related text files are listed below. Any parameter passed to the `saveMsObject()` through `...` are passed to the `saveMsObject()` calls of the individual MS data object(s) within the `XcmsExperiment`.

- The chromatographic peak information obtained with `chromPeaks()` and `chromPeaksData()` is stored in tabular format in the text files *xcms_experiment_chrom_peaks.txt* and *xcms_experiment_chrom_peak_data.tx* respectively. The first file's rows represent single peaks with their respective metadata in the columns (only numeric information). The second file contains arbitrary additional information/metadata for each peak (each row being one chrom peak).
- The `featureDefinitions()` are stored in a text file named *xcms_experiment_feature_definitions.txt*. Additionally, a second file named *ms_experiment_feature_peak_index.txt* is generated to connect the features with the corresponding chromatographic peaks. Each row of the first file corresponds to a feature with its respective metadata in the columns. The second file contains the mapping between features and chromatographic peaks (one peak ID per row).
- The `processHistory()` information of the object is stored to a file named *xcms_experiment_process_history.json* in JSON format.
- The `XcmsExperiment` directly extends the `MsExperiment` class, thus, any MS data is saved using a call to the `saveMsObject` of the `MsExperiment` (see above for more information).

Author(s)

Philippine Louail

See Also

Other MS object export and import formats.: `AlabasterParam`, `mzTabParam`

Examples

```
## Export and import a `Spectra` object:

library(Spectra)
library(msdata)
fl <- system.file("TripleTOF-SWATH", "PestMix1_DDA.mzML", package = "msdata")
sps <- Spectra(fl)

## Export the object to a temporary directory
d <- file.path(tempdir(), "spectra_example")
saveMsObject(sps, PlainTextParam(d))

## List the exported plain text files:
dir(d)

## - ms_backend_data.txt contains the metadata for the MS backend used (a
##   'MsBackendMzR`.
## - spectra_slots.txt contains general information from the Spectra object.

## Import the data again. By using `Spectra()` as first parameter we ensure
## the result is returned as a `Spectra` object.
```

```
sps_in <- readMsObject(Spectra(), PlainTextParam(d))
sps_in

## Check that the data is the same
all.equal(rtime(sps), rtime(sps_in))
all.equal(intensity(sps), intensity(sps_in))

## The data got exported *by module*, thus we could also load only a part of
## the exported data, such as just the `MsBackend` used by the `Spectra`:
be <- readMsObject(MsBackendMzR(), PlainTextParam(d))
be

## The export functionality also ensures that the data/object can be
## completely restored, i.e., for `Spectra` objects also their
## *processing queue* is preserved/stored. To show this we below first
## filter the spectra object by retention time and m/z:

sps_filt <- sps |>
    filterRt(c(400, 600)) |>
    filterMzRange(c(200, 300))
## The filtered object has less spectra
length(sps_filt)
length(sps)
## And also less mass peaks per spectrum
lengths(sps_filt[1:3])
lengths(sps[1:3])

d <- file.path(tempdir(), "spectra_example2")
saveMsObject(sps_filt, PlainTextParam(d))

## The directory contains now an additional file with the processing
## queue of the `Spectra`.
dir(d)

## Restoring the object again.
sps_in <- readMsObject(Spectra(), PlainTextParam(d))

## Both objects have the same processing history
sps_filt
sps_in

## Same number of spectra
length(sps_filt)
length(sps_in)

## Same number of mass peaks (after filtering)
lengths(sps_filt[1:3])
lengths(sps_in[1:3])
```

`saveMsObject` *Save and load MS data objects to and from different file formats*

Description

The `saveMsObject()` and `readMsObject()` methods allow serializing and restoring/importing mass spectrometry (MS) data objects to and from language agnostic file formats. The type and configuration of the file format is defined by the second argument to the method, `param`.

- `saveMsObject(object, param)`: saves the MS data object `object` to file(s) in a format defined by `param`.
- `readMsObject(object, param)`: `object` defines the type of MS object that should be returned by the function and `param` the format and file name(s) from which the data should be restored/imported.

Usage

```
saveMsObject(object, param, ...)

readMsObject(object, param, ...)
```

Arguments

`object`	for `saveMsObject()`: the MS data object to save, for `readMsObject()`: the MS data object that should be returned
`param`	an object defining and (eventually configuring) the file format and file name or directory to/from which the data object should be exported/imported.
`...`	additional optional arguments. See documentation of respective method for more information.

Value

`saveMsObject()` has no return value, `readMsObject` is expected to return an instance of the class defined with `object`.

Author(s)

Philippine Louail, Johannes Rainer, Laurent Gatto

Index