Package 'MsExperiment'

November 12, 2026

Title Infrastructure for Mass Spectrometry Experiments

Version 1.14.0

Description Infrastructure to store and manage all aspects related to a complete proteomics or metabolomics mass spectrometry (MS) experiment. The MsExperiment package provides light-weight and flexible containers for MS experiments building on the new MS infrastructure provided by the Spectra, QFeatures and related packages. Along with raw data representations, links to original data files and sample annotations, additional metadata or annotations can also be stored within the MsExperiment container. To guarantee maximum flexibility only minimal constraints are put on the type and content of the data within the containers.

Depends R (>= 4.2), ProtGenerics (>= 1.35.2),

Imports methods, S4Vectors, IRanges, Spectra, SummarizedExperiment, QFeatures, DBI, BiocGenerics

Suggests testthat, knitr (>= 1.1.0), roxygen2, BiocStyle (>= 2.5.19), rmarkdown, rpx, mzR, MsDataHub, MsBackendSql (>= 1.3.2), RSQLite

License Artistic-2.0

VignetteBuilder knitr

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

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

biocViews Infrastructure, Proteomics, MassSpectrometry, Metabolomics, ExperimentalDesign, DataImport

RoxygenNote 7.3.3

Roxygen list(markdown=TRUE)

Encoding UTF-8

Collate 'MsExperiment-db.R' 'MsExperiment-functions.R' 'MsExperimentFiles.R' 'MsExperiment.R' 'existMsExperimentFiles.R'

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

git_branch RELEASE_3_23

git_last_commit f47ab3c

git_last_commit_date 2026-04-28

Repository Bioconductor 3.23

Date/Publication 2026-11-11

Author Laurent Gatto [aut, cre] (ORCID:
<https://orcid.org/0000-0002-1520-2268>),
Johannes Rainer [aut] (ORCID: <https://orcid.org/0000-0002-6977-7147>),
Sebastian Gibb [aut] (ORCID: <https://orcid.org/0000-0001-7406-4443>),
Tuomas Borman [ctb] (ORCID: <https://orcid.org/0000-0002-8563-8884>)

Maintainer Laurent Gatto <laurent.gatto@uclouvain.be>

Contents

dbWriteSampleData 2
experimentFiles 4
MsExperimentFiles 11
readMsExperiment 12

Index **14**

`dbWriteSampleData` *Write sample annotations to a MsBackendSql SQL database*

Description

For `MsExperiment` objects with their MS data represented by a `Spectra` object that use a `MsBackendSql` backend, its sample annotations can be written to the backend's SQL database with the `dbWriteSampleData()` function. The content of the object's `[sampleData()]` (as well as eventually present *linking* between samples and spectra) will be stored in two separate database tables *sample_data* and *sample_to_msms_spectrum* in the same database.

This requires that the MS data of the experiment is *represented* by a `MsBackendSql` backend (see help on the `createMsBackendSqlDatabase` or the *MsBackendSql* package vignette for more information on how to create or use such SQL databases).

Usage

```
dbWriteSampleData(x)
```

Arguments

`x`	`MsExperiment` from which sample annotations should be written to the database.

Author(s)

Johannes Rainer, Laurent Gatto

Examples

```
library(MsExperiment)

## Create a MsBackendSql database from two mzML files.
## Connect first to an empty SQLite database (for the example we create
## a database in a temporary file).
library(RSQLite)
sqlite_db <- tempfile()
con <- dbConnect(SQLite(), sqlite_db)

## Define the files from which we import the data. Test files provided from
## the *MsDataHub*
fls <- c(MsDataHub::X20171016_POOL_POS_1_105.134.mzML(),
    MsDataHub::X20171016_POOL_POS_3_105.134.mzML())

## Create a MsBackendSql database containing the full MS data
library(MsBackendSql)
createMsBackendSqlDatabase(con, fls)

## Note: alternatively it would be possible to first import the MS data
## to a `Spectra` object and then change the backend to a `MsBackendSql`
## using the `setBackend` function.

## Load this data as a `Spectra` object (using a `MsBackendOfflineSql`
## backend)
library(Spectra)
sps <- Spectra(sqlite_db, source = MsBackendOfflineSql(),
    drv = SQLite())
sps

## Define sample annotations for the two data files. Adding one column
## `"file"` that contains the file name of the data files.
df <- data.frame(sample = c("QC1", "QC2"), file = basename(fls))

## Add a spectra variable `"file"` to the `Spectra` object with
## the raw data files' file names to simplify the linking between
## samples and spectra performed later.
sps$file <- basename(dataOrigin(sps))

## Create a MsExperiment with the spectra and sample data.
mse <- MsExperiment(spectra = sps, sampleData = df)

## Establish the link (mapping) between samples and spectra
## using the column `"file"` in the `sampleData` and the spectra
## variable `"file"`.
mse <- linkSampleData(mse, with = "sampleData.file = spectra.file")
mse
```

```
## Write sample data (and the sample to spectra mapping) to the
## *MsBackendSql* database.
dbWriteSampleData(mse)

## List the tables in the database
dbListTables(con)

## Sample data was thus stored to the database.
dbGetQuery(con, "select * from sample_data;")
```

`experimentFiles` *Managing Mass Spectrometry Experiments*

Description

The `MsExperiment` class allows the storage and management of all aspects related to a complete proteomics or metabolomics mass spectrometry experiment. This includes experimantal design (i.e. a table with samples), raw mass spectromtry data as spectra and chromatograms, quantitative features, and identification data or any other relevant data files.

For details, see https://rformassspectrometry.github.io/MsExperiment

This package is part of the RforMassSpectrometry initiative: https://www.rformassspectrometry.org/

Usage

```
experimentFiles(object)

experimentFiles(object) <- value

sampleData(object)

sampleData(object) <- value

qdata(object)

qdata(object) <- value

spectraSampleIndex(x, duplicates = c("first", "keep"))

MsExperiment(
  experimentFiles = MsExperimentFiles(),
  otherData = List(),
  qdata = NULL,
  sampleData = DataFrame(),
  spectra = NULL
)

## S4 method for signature 'MsExperiment'
```

```
show(object)

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

## S4 method for signature 'MsExperiment'
spectra(object)

## S4 replacement method for signature 'MsExperiment'
spectra(object) <- value

otherData(object)

otherData(object) <- value

linkSampleData(
  object,
  with = character(),
  sampleIndex = seq_len(nrow(sampleData(object))),
  withIndex = integer(),
  subsetBy = 1L
)

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

## S4 method for signature 'MsExperiment,function'
filterSpectra(object, filter, ...)
```

Arguments

object	An instance of class MsExperiment.
value	An object of the appropriate class for the slot to be populated.
x	an MsExperiment.
duplicates	for spectraSampleIndex(): character(1) defining the type of result returned by spectraSampleIndex(). With duplicates = "first" an integer vector is returned with the first match while duplicates = "keep" returns a list of integer with the index of all matches.
experimentFiles	MsExperimentFiles() defining (external) files to data or annotation.
otherData	List with arbitrary additional (*other*) information or data.
qdata	QFeatures or SummarizedExperiment with the quantification data.
sampleData	DataFrame (or data.frame) with information on individual samples of the experiment.
spectra	Spectra::Spectra() object with the MS spectra data of the experiment.

with	for linkSampleData(): character(1) defining the data to which samples should be linked. See section *Linking sample data to other experimental data* for details.
sampleIndex	for linkSampleData(): integer with the indices of the samples in sampleData(object) that should be linked.
withIndex	for linkSampleData(): integer with the indices of the elements in with to which the samples (specified by sampleIndex) should be linked to.
subsetBy	for linkSampleData(): optional integer(1) defining the dimension on which the subsetting will occurr on the linked data. Defaults to subsetBy = 1L thus subsetting will happen on the first dimension (rows or elements).
i	for [: an integer, character or logical referring to the indices or names (rowname of sampleData) of the samples to subset.
j	for [: not supported.
...	optional additional parameters. For filterSpectra(): parameters to be passed to the filter function (parameter filter).
drop	for [: ignored.
filter	for filterSpectra(): any filter function supported by Spectra::Spectra() to filter the spectra object (such as filterRt or filterMsLevel). Parameters for the filter function can be passed through ....

Value

See help of the individual functions.

Slots

`experimentFiles` An instance of class `MsExperimentFiles` or `NULL`.

`spectra` An instance of class `Spectra` or `NULL`.

`qdata` An instance of class `QFeatures`, `SummarizedExperiment` or `NULL`.

`otherData` A `List` to store any additional data objects.

`sampleData` A `DataFrame` documenting the experimental design.

`sampleDataLinks` A `List` with link definitions between samples and data elements. Should not be directly accessed or modified by the user.

`metadata` A `list` to store additional metadata.

General information

An experiment is typically composed of several items

- Description and information (covariates etc) of each sample from the experiment. These are stored in the `sampleData` slot as a `DataFrame`, each row describing a sample with columns containing all relevant information on that sample.
- Files to data or annotations. These are stored in the `@experimentFiles` slot as an instance of class `MsExperimentFiles`.
- General metadata about the experiment, stored as a `list` in the `@metadata` slot.

- Mass spectrometry data. Sectra and their metadata are stored as an [Spectra::Spectra()] object in the spectra slot. Chromatographic data is not yet supported but will be stored as a Chromatograms() object in the @chromatorgrams slot.
- Quantification data is stored as QFeatures or SummarizedExperiment objects in the @qdata slot and can be accessed or replaced with the qdata() or qdata<- functions, respectively.
- Any additional data, be it other spectra data, or proteomics identification data (i.e peptide-spectrum matches defined as PSM objects) can be added as elements to the list stored in the otherData slot.

The *length* of a MsExperiment is defined by the number of samples (i.e. the number of rows of the object's sampleData). A MsExperiment with two samples will thus have a length of two, independently of the number of files or length of raw data in the object. This also defines the subsetting of the object using the [function which will always subset by samples. See the section for filtering and subsetting below for more information.

MsExperiment objects can be created using the MsExperiment() function providing the data with the parameters listed below. If the Spectra::Spectra() object provided with the spectra param uses a MsBackendSql backend, sample data could be retrieved from the associated SQL database (see section *Using* MsExperiment *with* MsBackendSql in the vignette for details). Alternatively, it is also possible to subsequently add data and information to an existing MsExperiment. Finally, with the readMsExperiment() function it is possible to create a MsExperiment by importing MS spectra data directly from provided data files. See examples below or the package vignette for more information.

Accessing data

Data from an MsExperiment object can be accessed with the dedicated accessor functions:

- experimentFiles(), experimentFiles<-: gets or sets experiment files.
- length(): get the *length* of the object which represents the number of samples availble in the object's sampleData.
- metadata(), metadata<-: gets or sets the object's metadata.
- sampleData(), sampleData<-: gets or sets the object's sample data (i.e. a DataFrame containing sample descriptions).
- spectra(), spectra<-: gets or sets spectra data. spectra() returns a Spectra::Spectra() object, spectra<- takes a Spectra data as input and returns the updated MsExperiment.
- spectraSampleIndex(): depending on parameter duplicates it returns either an integer (duplicates = "first", the default) or a list (duplicates = "keep") of length equal to the number of spectra within the object with the indices of the sample(s) (in sampleData()) a spectrum is assigned to. With duplicates = "first", an integer with the index is returned for each spectrum. If a spectrum was assigned to more than one sample a warning is shown and only the first sample index is returned for that spectrum. For duplicates = "keep", assignments are returned as a list of integer vectors, each element being the index(es) of the sample(s) a spectrum is assigned to. For spectra that are not linked to any sample an NA_integer_ is returned as index for duplicates = "first" and an empty integer (integer()) for duplicates = "keep". Note that the default duplicates = "first" will work in almost all use cases, as generally, a spectrum will be assigned to a single sample.
- qdata(), qdata<-: gets or sets the quantification data, which can be a QFeatures or SummarizedExperiment.

- otherData() , otherData<-: gets or sets the addition data types, stored as a List in the object's otherData slot.

Linking sample data to other experimental data

To start with, an MsExperiment is just a loose collection of files and data related to an experiment, no explicit links or associactions are present between the samples and related data. Such links can however be created with the linkSampleData() function. This function can establish links between individual (or all) samples within the object's sampleData to individual, or multiple, data elements or files, such as Spectra or raw data files.

The presence of such links enables a (consistent) subsetting of an MsExperiment by samples. Thus, once the link is defined, any subsetting by sample will also correctly subset the linked data. All other, not linked, data elements are always retained as in the original MsExperiment.

To be able to link different elements within an MsExperiment it is also required to *identify* them with a consistent naming scheme. The naming scheme of slots and data elements within follows an SQL-like scheme, in which the variable (element) is identified by the name of the database table, followed by a "." and the name of the database table column. For MsExperiment, the naming scheme is defined as "<slot name>.<element name>". A column called "sample_name" within the sampleData data frame can thus be addressed with "sampleData.sample_name", while spectra.msLevel would represent the spectra variable called msLevel within the Spectra stored in the spectra slot.

Links between sample data rows and any other data element are stored as integer matrices within the @sampleDataLinks slot of the object (see also the vignette for examples and illustrations). The first column of a matrix is always the index of the sample, and the second column the index of the element that is linked to that sample, with one row per element. Links can be defined/added with the linkSampleData() function which adds a relationship between rows in sampleData to elements in any other data within the MsExperiment that are specified with parameter with. linkSampleData() supports two different ways to define the link:

- Parameter with defines the data to which the link should be established. To link samples to raw data files that would for example be available as a character in an element called "raw_files" within the object's experimentFiles, with = experimentFiles.raw_files would have to be used. Next it is required to specify which samples should be linked with which elements in with. This needs to be defined with the parameters sampleIndex and withIndex, both are expected to be integer vectors specifying which sample in sampleData should be linked to which element in with (see examples below or vignette for examples and details).
- As an alternative way, a link could be defined with an SQL-like syntax that relates a column in sampleData to a column/element in the data to which the link should be established. To link for example individual spectra to the corresponding samples with = "sampleData.raw_file = spectra.dataOrigin" could be used assuming that sampleData contains a column named "raw_file" with the (full path) of the raw data file for each sample from which the spectra were imported. In this case both sampleIndex and withIndex can be omitted, but it is expected/required that the columns/elements from sampleData and the data element to which the link should be established contain matching values.

Note that linkSampleData will **replace** a previously existing link to the same data element.

- `spectraSampleIndex()` is a convenience function that extracts for each spectrum in the object's `spectra()` the index of the sample it is associated with (see function's help above for more information).

Subsetting and filtering

- `[`: `MsExperiment` objects can be subset **by samples** with `[i]` where `i` is the index or a logical defining to which samples the data should be subset. Subsetting by sample will (correctly) subset all linked data to the respective samples. If multiple samples are linked to the same data element, subsetting might duplicate that data element. This duplication of *n:m* relationships between samples to elements does however not affect data consistency (see examples below for more information). Not linked data (slots) will be returned as they are. Subsetting in arbitrary order is supported. See the vignette for details and examples.
- `filterSpectra()`: subsets the `Spectra` within an `MsExperiment` using a provided filter function (parameter `filter`). Parameters for the filter function can be passed with parameter `...`. Any of the filter functions of a `Spectra::Spectra()` object can be passed with parameter `filter`. Possibly present relationships between samples and spectra (*links*, see also `linkSampleData()`) are updated. Filtering affects only the spectra data of the object, none of the other slots and data (e.g. `sampleData`) are modified. The function returns an `MsExperiment` with the filtered `Spectra` object.

Author(s)

Laurent Gatto, Johannes Rainer

Examples

```
## An empty MsExperiment object
msexp <- MsExperiment()
msexp

example(MsExperimentFiles)
experimentFiles(msexp) <- fls
msexp

## Linking samples to data elements

## Create a small experiment
library(S4Vectors)
mse <- MsExperiment()
sd <- DataFrame(sample_id = c("QC1", "QC2"),
                sample_name = c("QC Pool", "QC Pool"),
                injection_idx = c(1, 3))
sampleData(mse) <- sd

## define file names containing spectra data for the samples and
## add them, along with other arbitrary files to the experiment
fls <- c(MsDataHub::X20171016_POOL_POS_1_105.134.mzML(),
         MsDataHub::X20171016_POOL_POS_3_105.134.mzML())
experimentFiles(mse) <- MsExperimentFiles(
    mzML_files = fls,
```

```
    annotations = "internal_standards.txt")

## Link samples to data files: first sample to first file in "mzML_files",
## second sample to second file in "mzML_files"
mse <- linkSampleData(mse, with = "experimentFiles.mzML_files",
    sampleIndex = c(1, 2), withIndex = c(1, 2))

## Link all samples to the one file in "annotations"
mse <- linkSampleData(mse, with = "experimentFiles.annotations",
    sampleIndex = c(1, 2), withIndex = c(1, 1))
mse

## Import the spectra data and add it to the experiment
library(Spectra)
spectra(mse) <- Spectra(fls, backend = MsBackendMzR())

## Link each spectrum to the respective sample. We use the alternative
## link definition that does not require sampleIndex and withIndex but
## links elements based on matching values in the specified data elements.
## We need to add the full file name as an additional column to sampleData
## in order to allow matching this file names with the value in
## spectra(mse)$dataOrigin which contains the original file names from which
## the spectra were imported.
sampleData(mse)$raw_file <- normalizePath(fls)

## The links can be added using the short notation below
mse <- linkSampleData(mse, with = "sampleData.raw_file = spectra.dataOrigin")
mse

## With sampleData links present, any subsetting of the experiment by sample
## will ensure that all linked elements are subset accordingly
b <- mse[2]
b
sampleData(b)
experimentFiles(b)$mzML_files

## The `spectraSampleIndex()` function returns, for each spectrum, the
## index in the object's `sampleData` to which it is linked/assigned
spectraSampleIndex(mse)

## Subsetting with duplication of n:m sample to data relationships
##
## Both samples were assigned above to one "annotation" file in
## `experimentFiles`:
experimentFiles(mse[1])[["annotations"]]
experimentFiles(mse[2])[["annotations"]]

## Subsetting will always keep the relationship between samples and linked
## data elements. Subsetting will however possibly duplicate data elements
## that are shared among samples. Thus, while in the original object the
## element "annotations" has a single entry, subsetting with [1:2] will
## result in an MsExperiment with duplicated entries in "annotations"
experimentFiles(mse)[["annotations"]]
```

```
experimentFiles(mse[1:2])[["annotations"]]

## Spectra within an MsExperiment can be filtered/subset with the
## `filterSpectra` function and any of the filter functions supported
## by `Spectra` objects. Below we restrict the spectra data to spectra
## with a retention time between 200 and 210 seconds.
res <- filterSpectra(mse, filterRt, rt = c(200, 210))
res

## The object contains now much less spectra. The retention times for these
rtime(spectra(res))

## Relationship between samples and spectra was preserved by the filtering
a <- res[1L]
spectra(a)
```

`MsExperimentFiles` *A class to store experiment files*

Description

The `MsExperimentFiles` class stores files that are part of a mass spectrometry experiment. The objects are created with the `MsExperimentFiles()` function.

The files encoded in a `MsExperimentFiles` instance don't need to exist on the current filesystem - sometimes, these might be created in anticipation of their creation. The `existMsExperimentFiles()` function can be used to verify which ones currently exist: it returns a list of logicals (formally an instance of `IRanges::LogicalList()` of lenghts equal to the `MsExperimentFiles` used as input.

Usage

```
MsExperimentFiles(..., metadata = list())

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

existMsExperimentFiles(object)
```

Arguments

`...`	Either a named list or a set of named vectors. All elements are coerced to characters.
`metadata`	`list()` holding arbitrary R objects as annotations.
`object`	The `existMsExperimentFiles()` function works with either an instance of `MsExperimentFiles` or `MsExperiment`.

Value

`MsExperimentFiles` returns an instance of `MsExperimentFiles`.

Author(s)

Laurent Gatto

Examples

```
fls <- MsExperimentFiles(mzmls = c("/path/to/f1.mzML", "/path/to/f2.mzML"),
                         mzids = "/another/path/to/id1.mzid",
                         fasta = "file.fas")
fls

## A new MsExperimentFiles containing mzML or mzid files
fls[1]
fls["mzids"]

## The actual file names
fls[[1]]
fls[[2]]
fls[["fasta"]]

## None of the files used in this example actually exist
existMsExperimentFiles(fls)
```

---

readMsExperiment *Import MS spectra data of an experiment*

Description

Read/import MS spectra data of an experiment from the respective (raw) data files into an `MsExperiment()` object. Files provided with the `spectraFiles` parameter are imported as a `Spectra` object and each file is automatically *linked* to rows (samples) of a `sampleData` data frame (if provided).

Usage

```
readMsExperiment(spectraFiles = character(), sampleData = data.frame(), ...)
```

Arguments

`spectraFiles`	`character` with the (absolute) file names of the MS data files that should be imported as a `Spectra::Spectra()` object.
`sampleData`	`data.frame` or `DataFrame` with the sample annotations. Each row is expected to contain annotations for one file (sample). The order of the data frame's rows is expected to match the order of the provided files (with parameter `spectraFiles`).
`...`	additional parameters for the `Spectra::Spectra()` call to import the data.

Value

`MsExperiment`.

Author(s)

Johannes Rainer

Examples

```
## Define the files of the experiment to import. For our example we get
## the files from the *MsDataHub*
library(MsDataHub)
fls <- c(MsDataHub::X20171016_POOL_POS_1_105.134.mzML(),
         MsDataHub::X20171016_POOL_POS_3_105.134.mzML())

## Define a data frame with some sample annotations
ann <- data.frame(
    injection_index = c(1, 9),
    sample_id = c("QC_1", "QC_3"))

## Import the data
library(MsExperiment)
mse <- readMsExperiment(spectraFiles = fls, ann)
mse

## Access the spectra data
spectra(mse)

## Access the sample annotations
sampleData(mse)

## Import the data reading all MS spectra directly into memory
mse <- readMsExperiment(spectraFiles = fls, ann,
    backend = Spectra::MsBackendMemory())
mse
```

Index

[,MsExperiment,ANY,ANY,ANY-method
(experimentFiles), 4

dbWriteSampleData, 2

existMsExperimentFiles
(MsExperimentFiles), 11
experimentFiles, 4
experimentFiles<- (experimentFiles), 4

filterSpectra,MsExperiment,function-method
(experimentFiles), 4

IRanges::LogicalList(), *11*

length,MsExperiment-method
(experimentFiles), 4
linkSampleData (experimentFiles), 4

MsExperiment (experimentFiles), 4
MsExperiment(), *12*
MsExperiment-class (experimentFiles), 4
MsExperimentFiles, 11
MsExperimentFiles(), *5*
MsExperimentFiles-class
(MsExperimentFiles), 11

otherData (experimentFiles), 4
otherData<- (experimentFiles), 4

qdata (experimentFiles), 4
qdata<- (experimentFiles), 4

readMsExperiment, 12
readMsExperiment(), *7*

sampleData (experimentFiles), 4
sampleData<- (experimentFiles), 4
show,MsExperiment-method
(experimentFiles), 4
show,MsExperimentFiles-method
(MsExperimentFiles), 11
spectra,MsExperiment-method
(experimentFiles), 4
Spectra::Spectra(), *5–7*, *9*, *12*
spectra<-,MsExperiment-method
(experimentFiles), 4
spectraSampleIndex (experimentFiles), 4