Package ‘mzR’

February 21, 2019

Type Package

Title parser for netCDF, mzXML, mzData and mzML and mzIdentML files (mass spectrometry data)

Version 2.16.1

Author Bernd Fischer, Steffen Neumann, Laurent Gatto, Qiang Kou, Johannes Rainer

Maintainer Steffen Neumann <`sneumann@ipb-halle.de`>,
Laurent Gatto <`lg390@cam.ac.uk`>,
Qiang Kou <`qkou@umail.iu.edu`>

Description mzR provides a unified API to the common file formats and parsers available for mass spectrometry data. It comes with a wrapper for the ISB random access parser for mass spectrometry mzXML, mzData and mzML files. The package contains the original code written by the ISB, and a subset of the proteowizard library for mzML and mzIdentML. The netCDF reading code has previously been used in XCMS.

License Artistic-2.0

LazyLoad yes

Depends Rcpp (>= 0.10.1), methods, utils

Imports Biobase, BiocGenerics (>= 0.13.6), ProtGenerics (>= 1.9.1), ncdf4

Suggests msdata (>= 0.15.1), RUnit, mzID, BiocStyle (>= 2.5.19), knitr, XML

VignetteBuilder knitr

LinkingTo Rcpp, zlibbioc, Rhdf5lib (>= 1.1.4)

RcppModules Ramp, Pwiz, Ident

SystemRequirements C++11, GNU make

URL `https://github.com/sneumann/mzR/`

BugReports `https://github.com/sneumann/mzR/issues/`

biocViews ImmunoOncology, Infrastructure, DataImport, Proteomics, Metabolomics, MassSpectrometry

RoxygenNote 6.0.1

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

git_branch RELEASE_3_8

git_last_commit 94056b5

git_last_commit_date 2019-01-04

Date/Publication 2019-02-20

R topics documented:

mzR-package 2
copyWriteMSData 3
isolationWindow-methods 4
metadata 5
mzR-class 6
openMSfile 9
peaks 10
pwiz.version 13
writeMSData 13

Index **15**

`mzR-package` *parser for mzXML, mzData, mzML and mzid files (mass spectrometry data)*

Description

The mzR package is a library purely for accessing mass spectrometry data in a wide range of formats. Several backend libraries are used, such as the ISB random acces parser (RAMP) and ProteoWizard (pwiz) for mass spectrometry mzXML, mzData, mzML and mzid files. The package contains the original RAMP code written by the ISB, and a subset of the proteowizard library for mzML and mzid.

Details

Further information is available in the following vignette:

`mzR`	mzR, Ramp, mzXML, mzData, mzML (source, pdf)

Author(s)

Bernd Fischer, Steffen Neumann, Laurent Gatto, Qiang Kou

Maintainers: Bernd Fischer <bernd.fischer@embl.de>, Steffen Neumann <sneumann@ipb-halle.de>, Laurent Gatto <lg390@cam.ac.uk>, Qiang Kou <qkou@umail.iu.edu>

References

Nat Biotechnol. 2012 Oct 10;30(10):918-20. doi: 10.1038/nbt.2377. A cross-platform toolkit for mass spectrometry and proteomics. Chambers MC, Maclean B, Burke R, Amodei D, Ruderman DL, Neumann S, Gatto L, Fischer B, Pratt B, Egertson J, Hoff K, Kessner D, Tasman N, Shulman N, Frewen B, Baker TA, Brusniak MY, Paulse C, Creasy D, Flashner L, Kani K, Moulding C, Seymour SL, Nuwaysir LM, Lefebvre B, Kuhlmann F, Roark J, Rainer P, Detlev S, Hemenway T, Huhmer A, Langridge J, Connolly B, Chadick T, Holly K, Eckels J, Deutsch EW, Moritz RL, Katz JE, Agus DB, Maccoss M, Tabb DL, Mallick P. http://www.ncbi.nlm.nih.gov/pubmed/23051804

`copyWriteMSData` *Write MS spectrum data to a MS file copying metadata from the originating file*

Description

Copy general information from the originating MS file and write this, along with the provided spectra data, to a new file. The expected workflow is the following: data is first loaded from an MS file, e.g. using `peaks` and `header` methods, processed in R and then saved again to an MS file providing the (eventually) manipulated spectra and header data with arguments `header` and `data`.

Usage

```
copyWriteMSData(object, file, original_file, header, backend =
  "pwiz", outformat = "mzml", rtime_seconds = TRUE, software_processing)
```

Arguments

`object`	`list` containing for each spectrum one `matrix` with columns `mz` (first column) and `intensity` (second column). See also `peaks` for the method that reads such data from an MS file.
`file`	`character(1)` defining the name of the file.
`original_file`	`character(1)` with the name of the original file from which the spectrum data was first read.
`header`	`data.frame` with the header data for the spectra. Has to be in the format as the `data.frame` returned by the `header` method.
`backend`	`character(1)` defining the backend that should be used for writing. Currently only `"pwiz"` backend is supported.
`outformat`	`character(1)` the format of the output file. One of `"mzml"` or `"mzxml"`.
`rtime_seconds`	`logical(1)` whether the retention time is provided in seconds or minutes (defaults to `TRUE`).
`software_processing`	`list` of `character` vectors (or single `character` vector). Each `character` vector providing information about the software that was used to process the data with optional additional description of processing steps. The length of each `character` vector has to be >= 3: the first element being the name of the software, the second string its version and the third element the MS CV ID of the software (or `"MS:-1"` if not known). All additional elements are optional and represent the MS CV ID of each processing step performed with the software.

Note

`copyWriteMSData` supports at present copying data from `mzXML` and `mzML` and exporting to `mzML`. Copying and exporting to `mzXML` can fail for some input files.

The intention of this function is to copy data from an existing file and save it along with eventually modified data to a new file. To write new MS data files use the `writeMSData` function instead.

Author(s)

Johannes Rainer

See Also

`writeMSData` for a function to save MS data to a new mzML or mzXML file.

Examples

```
## Open a MS file and read the spectrum and header information
library(msdata)
fl <- system.file("threonine", "threonine_i2_e35_pH_tree.mzXML",
    package = "msdata")
ms_fl <- openMSfile(fl, backend = "pwiz")

## Get the spectra
pks <- spectra(ms_fl)
## Get the header
hdr <- header(ms_fl)

## Modify the spectrum data adding 100 to each intensity.
pks <- lapply(pks, function(z) {
    z[, 2] <- z[, 2] + 100
    z
})

## Copy metadata and additional information from the originating file
## and save it, along with the modified data, to a new mzML file.
out_file <- tempfile()
copyWriteMSData(pks, file = out_file, original_file = fl,
    header = hdr)
```

`isolationWindow-methods` *Returns the ion selection isolation window*

Description

The methods return matrices of lower (column `low`) and upper (column `high`) isolation window offsets. Matrices are returned as a list of length equal to the number of input files (provided as file names of raw mass spectrometry data objects, see below). By default (i.e when `unique. = TRUE`), only unique offsets are returned, as they are expected to identical for all spectra per acquisition. If this is not the case, a message is displayed.

Methods

`signature(object = "character", unique. = "logical", simplify = "logical")` Returns the isolation window for the file `object`. By default, only unique isolation windows are returned per file (`unique = TRUE`); if set to `FALSE`, a `matrix` with as many rows as there are MS2 spectra. If only one file passed an input and `simplify` is set to `TRUE` (default), the resulting `list` of length 1 is simplified to a `matrix`.

`signature(object = "mzRpwiz", unique. = "logical", simplify = "logical")` As above for `mzRpwiz` objects.

`signature(object = "mzRramp", unique. = "logical", simplify = "logical")` As above for `mzRramp` object.

Author(s)

Laurent Gatto <lg390@cam.ac.uk> based on the functionality from the `msPurity:::get_isolation_offsets` function.

Examples

```
library("msdata")
f <- msdata::proteomics(full.names = TRUE,
                    pattern = "TMT_Erwinia_1uLSike_Top10HCD_isol2_45stepped_60min_01.mzML.gz")
isolationWindow(f)

rw <- openMSfile(f)
isolationWindow(rw)
str(isolationWindow(rw, unique = FALSE))
```

`metadata` *Access the metadata from an* `mzR` *object.*

Description

Accessors to the analytical setup metadata of a run. `runInfo` will show a summary of the experiment as a named list, including `scanCount`, `lowMZ`, `highMZ`, `dStartTime`, `dEndTime` and `startTimeStamp`. Note that `startTimeStamp` can only be extracted from *mzML* files using the *pwiz* backend or from *CDF* files. A `NA` is reported if its value is not available. The `instrumentInfo` method returns a named `list` including instrument manufacturer, model, ionisation technique, analyzer and detector. `mzRpwiz` will give more additional information including information on sample, software using and original source file. These individual pieces of information can also be directly accessed by the specific methods. `mzidInfo` is used for the `mzR` object created from a mzid file. It returns basic information on this mzid file including file provider, creation date, software, database, enzymes and spectra data format. The `mzidInfo` will return the scoring results in identification. It will return different results for different searching software used.

Usage

```
runInfo(object)
chromatogramsInfo(object)
analyzer(object)
detector(object)
instrumentInfo(object)
ionisation(object)
softwareInfo(object)
sampleInfo(object)
sourceInfo(object)
model(object)
mzidInfo(object)
modifications(object, ...)
psms(object, ...)
substitutions(object)
```

```
database(object, ...)
enzymes(object)
tolerance(object)
score(x, ...)
para(object)
specParams(object)
```

Arguments

`object`	An instantiated `mzR` object.
`x`	An instantiated `mzR` object.
`...`	Additional arguments, currently ignored.

Author(s)

Steffen Neumann, Laurent Gatto and Qiang Kou

See Also

See for example `peaks` to access the data for the spectra in a `"mzR"` class.

Examples

```
library(msdata)

file <- system.file("microtofq/MM8.mzML", package = "msdata")
mz <- openMSfile(file)
fileName(mz)
instrumentInfo(mz)
runInfo(mz)
close(mz)

file <- system.file("cdf/ko15.CDF", package = "msdata")
mz <- openMSfile(file, backend = "netCDF")
fileName(mz)
instrumentInfo(mz)
runInfo(mz)
close(mz)

file <- system.file("mzid", "Tandem.mzid.gz", package="msdata")
mzid <- openIDfile(file)
softwareInfo(mzid)
enzymes(mzid)
```

`mzR-class` *Class* `mzR` *and sub-classes*

Description

The class `mzR` is the main class for the common mass spectrometry formats. It is a virtual class and thus not supposed to be instanciated directly.

The sub-classes implement specific APIs to access the underlying data and metadata in the files. Currently, `mzRramp` and `mzRpwiz` are available implementations. `mzRramp` uses the ISB 'RAMP' random access C/C++ API, and `mzRpwiz` uses Proteowizard to access the relevant information in `mzData`, `mzXML` and `mzML` files.

Additional sub-classes using the proteowizard API and netCDF are planned.

Objects from the Class

`mzR` is a virtual class, so instances cannot be created.

Objects can be created by calls of the form `new("mzRramp", ...)`, but more often they will be created with `openMSfile`.

After creating a `mzR`, you can write it into a file. mzXML, mzML, mgf formats are supported.

Slots

`fileName`: Object of class `character` storing the original filename used when the instance was created.

`backend`: One of the implemented backens or `NULL`.

`.__classVersion__`: Object of class `"Versioned"`, from Biobase.

Extends

Class `"Versioned"`, directly.

Methods

For methods to access raw data (spectra and chromatograms), see `peaks`.

Methods currently implemented for `mzR`

fileName `signature(object = "mzR")`: ...

Methods currently implemented for `mzRramp`

analyzer `signature(object = "mzRramp")`: ...

close `signature(con = "mzRramp")`: ...

detector `signature(object = "mzRramp")`: ...

fileName `signature(object = "mzRramp")`: ...

initializeRamp `signature(object = "mzRramp")`: ...

instrumentInfo `signature(object = "mzRramp")`: ...

ionisation `signature(object = "mzRramp")`: ...

isInitialized `signature(object = "mzRramp")`: ...

length `signature(x = "mzRramp")`: ...

manufacturer `signature(object = "mzRramp")`: ...

model `signature(object = "mzRramp")`: ...

runInfo `signature(object = "mzRramp")`: ...

Methods currently implemented for `mzRpwiz`

analyzer `signature(object = "mzRpwiz")`: ...

detector `signature(object = "mzRpwiz")`: ...

instrumentInfo `signature(object = "mzRpwiz")`: ...

ionisation `signature(object = "mzRpwiz")`: ...

length `signature(x = "mzRpwiz")`: ...

manufacturer `signature(object = "mzRpwiz")`: ...

model `signature(object = "mzRpwiz")`: ...

runInfo `signature(object = "mzRpwiz")`: ...

chromatogramsInfo `signature(object = "mzRpwiz")`: ...

Methods currently implemented for `mzRident`

mzidInfo `signature(object = "mzRident")`: ...

psms `signature(object = "mzRident")`: ...

modifications `signature(object = "mzRident")`: ...

substitutions `signature(object = "mzRident")`: ...

database `signature(x = "mzRident")`: ...

enzymes `signature(object = "mzRident")`: ...

sourceInfo `signature(object = "mzRident")`: ...

tolerance `signature(object = "mzRident")`: ...

score `signature(object = "mzRident")`: ...

para `signature(object = "mzRident")`: ...

specParams `signature(object = "mzRident")`: ...

Author(s)

Steffen Neumann, Laurent Gatto, Qiang Kou

References

RAMP: http://tools.proteomecenter.org/wiki/index.php?title=Software:RAMP Proteowizard: http://proteowizard.sourcef

Examples

```
library(msdata)
filepath <- system.file("microtofq", package = "msdata")
file <- list.files(filepath, pattern="MM14.mzML",
                   full.names=TRUE, recursive = TRUE)
mzml <- openMSfile(file)
close(mzml)

## using the pwiz backend
mzml <- openMSfile(file, backend = "pwiz")
```

`openMSfile` *Create and check mzR objects from netCDF, mzXML, mzData or mzML files.*

Description

The `openMSfile` constructor will create a new format-specifc `mzR` object, open 'filename' file and all header information is loaded as a Rcpp module and made accessible through the `ramp` or `pwiz` slot of the resulting object.

The `openIDfile` constructor will create a new format-specifc `mzR` object, open 'filename' file and all information is loaded as a Rcpp module. The mzid format is supported throught `pwiz` backend. Only mzIdentML 1.1 is supported.

Usage

```
openMSfile(filename, backend = NULL, verbose = FALSE)

initializeRamp(object)

isInitialized(object)

fileName(object, ...)

openIDfile(filename, verbose = FALSE)
```

Arguments

`filename`	Path name of the netCDF, mzData, mzXML or mzML file to read/write.
`backend`	A `character(1)` specifiying which backend API to use. Currently 'Ramp', 'netCDF' and 'pwiz' are supported. If `backend = NULL` (the default), the function tries to determine the backend to be used based on either the file extension of the file content.
`object`	An instantiated mzR object.
`verbose`	Enable verbose output.
`...`	Additional arguments, currently ignored.

Author(s)

Steffen Neumann, Laurent Gatto, Qiang Kou

Examples

```
library(msdata)
filepath <- system.file("microtofq", package = "msdata")
file <- list.files(filepath, pattern="MM14.mzML",
                   full.names=TRUE, recursive = TRUE)
mz <- openMSfile(file)
fileName(mz)
runInfo(mz)
close(mz)
```

```
 ## Not run:
    ## to use another backend
    mz <- openMSfile(file, backend = "pwiz")
    mz

## End(Not run)

  file <- system.file("mzid", "Tandem.mzid.gz", package="msdata")
  mzid <- openIDfile(file)
  softwareInfo(mzid)
  enzymes(mzid)
```

`peaks` *Access the raw data from an* `mzR` *object.*

Description

Access the MS raw data. The `peaks`, `spectra` (can be used interchangeably) and `peaksCount` functions return the (m/z, intensity) pairs and the number peaks in the spectrum/spectra. `peaks` and `spectra` return a single matrix if `scans` is a `numeric` of length 1 and a list of matrices if several scans are asked for or no `scans` argument is provided (i.e all spectra in the oject are retured). `peaksCount` will return a numeric of length `n`.

The `header` function returns a list containing `seqNum`, `acquisitionNum`, `msLevel`, `peaksCount`, `totIonCurrent`, `retentionTime` (in seconds), `basePeakMZ`, `basePeakIntensity`, `collisionEnergy`, `ionisationEnergy`, `lowM`, `highMZ`, `precursorScanNum`, `precursorMZ`, `precursorCharge`, `precursorIntensity`, `mergedScan`, `mergedResultScanNum`, `mergedResultStartScanNum`, `mergedResultEndScanNum`, `filterString`, `spectrumId`, `centroided` (`logical` whether the data of the spectrum is in centroid mode or profile mode; only for pwiz backend), `injectionTime` (ion injection time, in milliseconds) and `ionMobilityDriftTime` (in milliseconds). If multiple scans are queried, a `data.frame` is returned with the scans reported along the rows.

The `get3Dmap` function performs a simple resampling between `lowMz` and `highMz` with `reMz` resolution. A matrix of dimensions `length(scans)` times `seq(lowMz,highMz,resMz)` is returned.

The `chromatogram` (`chromatograms`) accessors return chromatograms for the MS file handle. If a single index is provided, as `data.frame` containing the retention time (1st columns) and intensities (2nd column) is returned. The name of the former is always `time`, while the latter will depend on the run parameters.

If more than 1 or no chromatogram indices are provided, then a list of chromatograms is returned; either those passed as argument or all of them. By default, the first (and possibly only) chromatogram is the total ion count, which can also be accessed with the `tic` method.

The `nChrom` function returns the number of chromatograms, including the total ion chromatogram.

The `chromatogramHeader` returns (similar to the `header` function for spectra) a `data.frame` with metadata information for the individual chromatograms. The `data.frame` has the columns: `"chromatogramId"` (the ID of the chromatogram as specified in the file), `"chromatogramIndex"` (the index of the chromatogram within the file), `"polarity"` (the polarity for the chromatogram, 0 for negative, +1 for positive and -1 for not set), `"precursorIsolationWindowTargetMZ"` (the isolation window m/z of the precursor), `"precursorIsolationWindowLowerOffset"`, `"precursorIsolationWindowUpperOffset"` (lower and upper offset for the isolation window m/z), `"precursorCollisionEnergy"` (collision energy), `"productIsolationWindowTargetMZ"`, `"productIsolationWindowLowerOffset"` and `"productIsolationWindowUpperOffset"` (definition of the m/z isolation window for the product).

Note that access to chromatograms is only supported in the `pwiz` backend.

Usage

```
 header(object, scans, ...)

 peaksCount(object, scans, ...)

 ## S4 method for signature 'mzRpwiz'
peaks(object, scans)
 ## S4 method for signature 'mzRramp'
peaks(object, scans)
 ## S4 method for signature 'mzRnetCDF'
peaks(object, scans)

 ## S4 method for signature 'mzRpwiz'
spectra(object, scans)   ## same as peaks
 ## S4 method for signature 'mzRramp'
spectra(object, scans)
 ## S4 method for signature 'mzRnetCDF'
spectra(object, scans)

 get3Dmap(object, scans, lowMz, highMz, resMz, ...)

 ## S4 method for signature 'mzRpwiz'
chromatogram(object, chrom)

 ## S4 method for signature 'mzRpwiz'
chromatograms(object, chrom)  ## same as chromatogram

 ## S4 method for signature 'mzRpwiz'
chromatogramHeader(object, chrom)

 tic(object, ...)

 nChrom(object)
```

Arguments

`object`	An instantiated `mzR` object.
`scans`	A `numeric` specifying which scans to return. Optional for the `header`, `peaks`, `spectra` and `peaksCount` methods. If ommited, the requested data for all peaks is returned.
`lowMz, highMz`	`Numerics` defining the m/z range to be returned.
`resMz`	a `numeric` defining the m/z resolution.
`chrom`	For `chromatogram`, `chromatograms` and `chromatogramHeader`: `numeric` specifying the index of the chromatograms to be extracted from the file. If omitted, data for all chromatograms is returned.

`...`	Other arguments. A `scan` parameter can be passed to `peaks`.

Details

The column `acquisitionNum` in the `data.frame` returned by the `header` method contains the index during the scan in which the signal from the spectrum was measured. The `pwiz` backend extracts this number from the spectrum's ID provided in the mzML file. In contrast, column `seqNum` contains the index of each spectrum within the file and is thus consecutively numbered. Spectra from files with multiple MS levels are linked to each other *via* their `acquisitionNum`: column `precursorScanNum` of an e.g. MS level 2 spectrum contains the `acquisitionNum` of the related MS level 1 spectrum.

Note

Spectrum identifiers are only specified in *mzML* files, thus, for all other file types the column `"spectrumId"` of the result `data.frame` returned by `header` contains `"scan="` followed by the acquisition number of the spectrum. Also, only the `pwiz` backend supports extraction of the spectras' IDs from *mzML* files. Thus, only *mzML* files read with `backend = "pwiz"` provide the spectrum IDs defined in the file. The content of the spectrum identifier depends on the vendor and the instrument acquisition settings and is reported here as a character, in its raw form, without further parsing.

Author(s)

Steffen Neumann and Laurent Gatto

See Also

`instrumentInfo` for metadata access and the `"mzR"` class.

`writeMSData` and `copyWriteMSData` for functions to write MS data in *mzML* or `mzXML` format.

Examples

```
library(msdata)
filepath <- system.file("microtofq", package = "msdata")
file <- list.files(filepath, pattern="MM14.mzML",
                   full.names=TRUE, recursive = TRUE)
mz <- openMSfile(file)
runInfo(mz)
colnames(header(mz))
close(mz)

## A shotgun LCMSMS experiment
f <- proteomics(full.names = TRUE,
             pattern = "TMT_Erwinia_1uLSike_Top10HCD_isol2_45stepped_60min_01.mzML.gz")
x <- openMSfile(f, backend = "pwiz")
x
nChrom(x)
head(tic(x))
head(chromatogram(x, 1L)) ## same as tic(x)
str(chromatogram(x)) ## as a list

p <- peaks(x) ## extract all peak information
head(peaks(x, scan=4)) ## extract just peaks from the 4th scan

## An MRM experiment
```

```
f <- proteomics(full.names = TRUE, pattern = "MRM")
x <- openMSfile(f, backend = "pwiz")
x
nChrom(x)
head(tic(x))
head(chromatogram(x, 1L)) ## same as tic(x)
str(chromatogram(x, 10:12))

## get the header information for the chromatograms
ch <- chromatogramHeader(x)
head(ch)
```

`pwiz.version` *Get the version number of pwiz backend.*

Description

Get the version number of pwiz backend.

Usage

```
pwiz.version()
```

`writeMSData` *Write MS spectrum data to an MS file*

Description

`writeMSData` exports the MS spectrum data provided with parameters `header` and `data` to an MS file in mzML or mzXML format.

Usage

```
## S4 method for signature 'list,character'
writeMSData(object, file, header,
    backend = "pwiz", outformat = "mzml", rtime_seconds = TRUE,
    software_processing)
```

Arguments

`object`	`list` containing for each spectrum one `matrix` with columns `mz` (first column) and `intensity` (second column). See also `peaks` for the method that reads such data from an MS file.
`file`	`character(1)` defining the name of the file.
`header`	`data.frame` with the header data for the spectra. Has to be in the format as the `data.frame` returned by the `header` method.
`backend`	`character(1)` defining the backend that should be used for writing. Currently only `"pwiz"` backend is supported.

`outformat`	`character(1)` the format of the output file. One of `"mzml"` or `"mzxml"`.
`rtime_seconds`	`logical(1)` whether the retention time is provided in seconds or minutes (defaults to `TRUE`).
`software_processing`	`list` of `character` vectors (or single `character` vector). Each `character` vector providing information about the software that was used to process the data with optional additional description of processing steps. The length of each `character` vector has to be >= 3: the first element being the name of the software, the second string its version and the third element the MS CV ID of the software (or `"MS:-1"` if not known). All additional elements are optional and represent the MS CV ID of each processing step performed with the software.

Author(s)

Johannes Rainer

See Also

`copyWriteMSData` for a function to copy general information from a MS data file and writing eventually modified MS data from that originating file.

Examples

```
## Open a MS file and read the spectrum and header information
library(msdata)
fl <- system.file("threonine", "threonine_i2_e35_pH_tree.mzXML",
    package = "msdata")
ms_fl <- openMSfile(fl, backend = "pwiz")

## Get the spectra
pks <- spectra(ms_fl)
## Get the header
hdr <- header(ms_fl)

## Modify the spectrum data adding 100 to each intensity.
pks <- lapply(pks, function(z) {
    z[, 2] <- z[, 2] + 100
    z
})

## Write the data to a mzML file.
out_file <- tempfile()
writeMSData(object = pks, file = out_file, header = hdr)
```

Index

∗Topic **classes**
    mzR-class, 6
∗Topic **methods**
    isolationWindow-methods, 4
∗Topic **package, file**
    mzR-package, 2

analyzer *(metadata)*, 5
analyzer,mzRnetCDF-method *(mzR-class)*, 6
analyzer,mzRpwiz-method *(mzR-class)*, 6
analyzer,mzRramp-method *(mzR-class)*, 6

chromatogram *(peaks)*, 10
chromatogram,mzRnetCDF-method *(peaks)*, 10
chromatogram,mzRpwiz-method *(peaks)*, 10
chromatogram,mzRramp-method *(peaks)*, 10
chromatogramHeader *(peaks)*, 10
chromatogramHeader,mzRnetCDF-method *(peaks)*, 10
chromatogramHeader,mzRpwiz-method *(peaks)*, 10
chromatogramHeader,mzRramp-method *(peaks)*, 10
chromatograms *(peaks)*, 10
chromatograms,mzRnetCDF-method *(peaks)*, 10
chromatograms,mzRpwiz-method *(peaks)*, 10
chromatograms,mzRramp-method *(peaks)*, 10
chromatogramsInfo *(metadata)*, 5
chromatogramsInfo,mzRpwiz-method *(mzR-class)*, 6
class:mzR *(mzR-class)*, 6
class:mzRident *(mzR-class)*, 6
class:mzRnetCDF *(mzR-class)*, 6
class:mzRpwiz *(mzR-class)*, 6
class:mzRramp *(mzR-class)*, 6
close *(mzR-class)*, 6
close,mzRnetCDF-method *(mzR-class)*, 6
close,mzRpwiz-method *(mzR-class)*, 6
close,mzRramp-method *(mzR-class)*, 6
copyWriteMSData, 3, *12*, *14*

database *(metadata)*, 5
database,mzRident-method *(mzR-class)*, 6
detector *(metadata)*, 5
detector,mzRnetCDF-method *(mzR-class)*, 6
detector,mzRpwiz-method *(mzR-class)*, 6
detector,mzRramp-method *(mzR-class)*, 6

enzymes *(metadata)*, 5
enzymes,mzRident-method *(mzR-class)*, 6

fileName *(openMSfile)*, 9
fileName,mzR-method *(mzR-class)*, 6

get3Dmap *(peaks)*, 10
get3Dmap,mzRpwiz-method *(peaks)*, 10
get3Dmap,mzRramp-method *(peaks)*, 10

header, *3*, *10*, *13*
header *(peaks)*, 10
header,mzRnetCDF,missing-method *(peaks)*, 10
header,mzRnetCDF,numeric-method *(peaks)*, 10
header,mzRpwiz,missing-method *(peaks)*, 10
header,mzRpwiz,numeric-method *(peaks)*, 10
header,mzRramp,missing-method *(peaks)*, 10
header,mzRramp,numeric-method *(peaks)*, 10

initializeRamp *(openMSfile)*, 9
initializeRamp,mzRramp-method *(mzR-class)*, 6
instrumentInfo, *12*
instrumentInfo *(metadata)*, 5
instrumentInfo,mzRnetCDF-method *(mzR-class)*, 6
instrumentInfo,mzRpwiz-method *(mzR-class)*, 6
instrumentInfo,mzRramp-method *(mzR-class)*, 6
ionisation *(metadata)*, 5
ionisation,mzRnetCDF-method *(mzR-class)*, 6

ionisation,mzRpwiz-method (mzR-class), 6
ionisation,mzRramp-method (mzR-class), 6
isInitialized (openMSfile), 9
isInitialized,mzRnetCDF-method (mzR-class), 6
isInitialized,mzRramp-method (mzR-class), 6
isolationWindow (isolationWindow-methods), 4
isolationWindow,character-method (isolationWindow-methods), 4
isolationWindow,mzRpwiz-method (isolationWindow-methods), 4
isolationWindow,mzRramp-method (isolationWindow-methods), 4
isolationWindow-methods, 4

length (mzR-class), 6
length,mzRident-method (mzR-class), 6
length,mzRnetCDF-method (mzR-class), 6
length,mzRpwiz-method (mzR-class), 6
length,mzRramp-method (mzR-class), 6

manufacturer (metadata), 5
manufacturer,mzRnetCDF-method (mzR-class), 6
manufacturer,mzRpwiz-method (mzR-class), 6
manufacturer,mzRramp-method (mzR-class), 6
metadata, 5
model (metadata), 5
model,mzRnetCDF-method (mzR-class), 6
model,mzRpwiz-method (mzR-class), 6
model,mzRramp-method (mzR-class), 6
modifications (metadata), 5
modifications,mzRident-method (mzR-class), 6
mzidInfo (metadata), 5
mzidInfo,mzRident-method (mzR-class), 6
mzR, *6*, *12*
mzR (mzR-package), 2
mzR-class, 6
mzR-package, 2
mzRident-class (mzR-class), 6
mzRnetCDF-class (mzR-class), 6
mzRpwiz-class (mzR-class), 6
mzRramp-class (mzR-class), 6

nChrom (peaks), 10

openIDfile (openMSfile), 9
openMSfile, *7*, 9

para (metadata), 5
para,mzRident-method (mzR-class), 6
peaks, *3*, *6*, *7*, 10, *13*
peaks,mzRnetCDF-method (peaks), 10
peaks,mzRpwiz-method (peaks), 10
peaks,mzRramp-method (peaks), 10
peaksCount (peaks), 10
peaksCount,mzRpwiz,missing-method (peaks), 10
peaksCount,mzRpwiz,numeric-method (peaks), 10
peaksCount,mzRramp,missing-method (peaks), 10
peaksCount,mzRramp,numeric-method (peaks), 10
psms (metadata), 5
psms,mzRident-method (mzR-class), 6
pwiz.version, 13

runInfo (metadata), 5
runInfo,mzRnetCDF-method (mzR-class), 6
runInfo,mzRpwiz-method (mzR-class), 6
runInfo,mzRramp-method (mzR-class), 6

sampleInfo (metadata), 5
sampleInfo,mzRpwiz-method (mzR-class), 6
score (metadata), 5
score,mzRident-method (mzR-class), 6
softwareInfo (metadata), 5
softwareInfo,mzRident-method (mzR-class), 6
softwareInfo,mzRpwiz-method (mzR-class), 6
sourceInfo (metadata), 5
sourceInfo,mzRident-method (mzR-class), 6
sourceInfo,mzRpwiz-method (mzR-class), 6
specParams (metadata), 5
specParams,mzRident-method (mzR-class), 6
spectra (peaks), 10
spectra,mzRnetCDF-method (peaks), 10
spectra,mzRpwiz-method (peaks), 10
spectra,mzRramp-method (peaks), 10
substitutions (metadata), 5
substitutions,mzRident-method (mzR-class), 6

tic (peaks), 10
tic,mzRpwiz-method (peaks), 10
tolerance (metadata), 5
tolerance,mzRident-method (mzR-class), 6

Versioned, *7*

`writeMSData`, *3*, *4*, *12*, 13
`writeMSData,list,character-method` (`writeMSData`), 13