Package: RaMS (via r-universe)

December 9, 2024

Type Package

Title R Access to Mass-Spec Data

Version 1.4.3

Maintainer William Kumler <wkumler@uw.edu>

Description R-based access to mass-spectrometry (MS) data. While many packages exist to process MS data, many of these make it difficult to access the underlying mass-to-charge ratio (m/z), intensity, and retention time of the files themselves. This package is designed to format MS data in a tidy fashion and allows the user perform the plotting and analysis.

License MIT + file LICENSE

URL https://github.com/wkumler/RaMS

BugReports https://github.com/wkumler/RaMS/issues

Encoding UTF-8

Imports xml2, base64enc, data.table, utils, stats

RoxygenNote 7.3.1

Suggests testthat, knitr, rmarkdown, tidyverse, ggplot2, dplyr, plotly, openxlsx, DBI, RSQLite, reticulate, BiocParallel, Spectra, arrow, microbenchmark, rvest

VignetteBuilder knitr

NeedsCompilation no

Author William Kumler [aut, cre, cph], Ricardo Cunha [ctb], Ethan Bass [ctb]

Repository CRAN

Date/Publication 2024-10-09 05:00:02 UTC

Config/pak/sysreqs libxml2-dev

Contents

`checkOutputQuality` *Check that the output data is properly formatted.*

Description

This function checks that data produced by repeated calls to the 'grabMzmlData()' and 'grabMzxmlData()' functions is formatted properly before it's provided to the user. It checks that all of the requested data has been obtained and warns if data is found to be empty, misnamed, or has columns of the wrong type.

Usage

```
checkOutputQuality(output_data, grab_what)
```

Arguments

`output_data`	The collected data resulting from repeated calls to 'grabMzmlData()', after being bound together.
`grab_what`	The names of the data requested by the user.

Value

NULL (invisibly). The goal of this function is its side effects, i.e. throwing errors and providing info when the files are not found.

`editMSfileRTs` *Edit mzML/mzXML file retention times*

Description

This function uses the basic XML parsing of RaMS to modify the retention times of MS scans within the mzML/mzXML files. This method can be useful for performing RT correction using one platform and then peakpicking and correspondence on another. The basic method is simply replacing the scan's recorded retention time value with an arbitrary one of your choosing. This function is vectorized to handle multiple files, while the internal 'editMzmlRTs()' and 'editMzxmlRTs()' do most of the heavy lifting. Note that the seconds vs minutes must be closely monitored here - the unit should be the same as the one in the file originally.

Usage

```
editMSfileRTs(
  files,
  new_rt_list,
  new_filenames = NULL,
  interp_method = "linear",
  overwrite = FALSE
)
```

Arguments

files	Vector of filenames (including the relative/absolute path)
new_rt_list	Nested of new retention times. One entry in the list for each file (in the same order as the files), each containing a vector of new retention times. RT vectors can be equal to either every scan or just every MS1 scan. If only the MS1 scans are provided in a file with additional MS levels, MSn scan RTs will be interpolated according to interp_method (below)
new_filenames	Vector of filenames (including relative/absolute paths) describing where the edited files should be written out. Can be the same as files but will throw a warning and append _rtcor to each file unless 'overwrite = TRUE' (below)
interp_method	Either "linear" or "constant". Describes the way that MSn retention times should be handled when only the MS1 values are provided. "linear" (the default) means that the spacing will be preserved, while "constant" will use the associated MS1 scan RT for all MSn scans, allowing an easy method of linking the MSn to the MS1.
overwrite	Boolean. Controls whether files are overwritten in place if 'new_filenames' is not provided.

Value

Invisibly, the names of the edited files.

Examples

```
## Not run:
# Setup (allows running on CRAN computers)
example_dir <- tempdir()
rams_dir <- system.file("extdata", package = "RaMS")
file.copy(list.files(rams_dir, pattern = "LB.*mzML", full.names = TRUE), example_dir)
mzMLs <- list.files(example_dir, pattern = "LB.*mzML", full.names = TRUE)

library(xcms)
library(RaMS)

register(SerialParam())
xcms_obj <- readMSData(mzMLs, msLevel. = 1, mode = "onDisk")
cwp <- CentWaveParam(ppm = 5, peakwidth = c(20, 80))
xcms_peakpicked <- findChromPeaks(xcms_obj, param = cwp)
xcms_rtcor <- adjustRtime(xcms_peakpicked, param = ObiwarpParam())

# Extract the adjusted RTs from the XCMS object
new_rts <- split(rtime(xcms_rtcor)/60, fromFile(xcms_rtcor))
# Apply the retention time correction to the new files
mzMLs_rtcor <- editMSfileRTs(mzMLs, new_rt_list = new_rts)

# Contrast the two chromatograms to see the peaks aligned
qplotMS1data(grabMSdata(mzMLs)$MS1[mz%between%pmppm(104.1073, 10)])
qplotMS1data(grabMSdata(mzMLs_rtcor)$MS1[mz%between%pmppm(104.1073, 10)])
```

```
# Cleanup
file.remove(mzMLs)
file.remove(mzMLs_rtcor)

## End(Not run)
```

`getEncoded` *Convert from compressed binary to R numeric vector*

Description

Convert from compressed binary to R numeric vector

Usage

```
getEncoded(mzint_nodes, compression_type, bin_precision, endi_enc)
```

Arguments

`mzint_nodes`	The XML nodes containing the compressed binary string
`compression_type`	Compression type to be used by memDecompress
`bin_precision`	The bit (?) precision used by readBin
`endi_enc`	The byte order (?) of the string. For mzML this is always "little" but mzXML can also be "big"

Value

A numeric vector of m/z or intensity values

`giveEncoding` *Convert from R numeric vector to compressed binary*

Description

Convert from R numeric vector to compressed binary

Usage

```
giveEncoding(mzint_vals, compression_type, bin_precision, endi_enc)
```

Arguments

mzint_vals	A numeric vector of m/z or intensity values
compression_type	Compression type to be used by memCompress
bin_precision	The bit (?) precision used by writeBin
endi_enc	The byte order (?) of the string. For mzML this is always "little" but mzXML can also be "big"

Value

A single base64-encoded string of compressed binary values

---

grabAccessionData *Get arbitrary metadata from an mzML file by accession number*

Description

Get arbitrary metadata from an mzML file by accession number

Usage

```
grabAccessionData(filename, accession_number)
```

Arguments

filename	The name of the file for which metadata is requested. Both absolute and relative paths are acceptable.
accession_number	The HUPO-PSI accession number for the metadata to be extracted. These accession numbers are typically of the form MS:######## and the full list can be found and searched at https://raw.githubusercontent.com/HUPO-PSI/psi-ms-CV/master/psi-ms.obo.

Value

A data frame with the name and value of the parameter requested, as deduced from the XML tag attributes corresponding to the accession number.

Examples

```
library(RaMS)

sample_dir <- system.file("extdata", package = "RaMS")
sample_file <- list.files(sample_dir, full.names=TRUE)[3]
# Get highest observed m/z detected
topmass_df <- grabAccessionData(sample_file, "MS:1000527")
# Manually create TIC
```

```
int_df <- grabAccessionData(sample_file, "MS:1000285")
rt_df <- grabAccessionData(sample_file, "MS:1000016")
tic <- data.frame(rt=rt_df$value, int=int_df$value)
plot(tic$rt, tic$int, type = "l")
```

`grabMSdata` *Grab mass-spectrometry data from file(s)*

Description

The main 'RaMS' function. This function accepts a list of the files that will be read into R's working memory and returns a list of 'data.table's containing the requested information. What information is requested is determined by the 'grab_what' argument, which can include MS1, MS2, BPC, TIC, or metadata information. This function serves as a wrapper around both 'grabMzmlData' and 'grabMzxmlData' and handles multiple files, but those two have also been exposed to the user in case super-simple handling is desired. Retention times are reported in minutes, and will be converted automatically if they are encoded in seconds.

Usage

```
grabMSdata(
  files,
  grab_what = "everything",
  verbosity = NULL,
  incl_polarity = FALSE,
  mz = NULL,
  ppm = NULL,
  rtrange = NULL,
  prefilter = -1
)
```

Arguments

`files`	A character vector of filenames to read into R's memory. Both absolute and relative paths are acceptable.
`grab_what`	What data should be read from the file? Options include "MS1" for data only from the first spectrometer, "MS2" for fragmentation data, "BPC" for rapid access to the base peak chromatogram, "TIC" for rapid access to the total ion chromatogram, "DAD" for DAD (UV) data, and "chroms" for precompiled chromatogram data (especially useful for MRM but often contains BPC/TIC in other files). Metadata can be accessed with "metadata", which provides information about the instrument and time the file was run. These options can be combined (i.e. 'grab_data=c("MS1", "MS2", "BPC")') or this argument can be set to "everything" to extract all of the above. Options "EIC" and "EIC_MS2" are useful when working with files whose total size exceeds working memory - it first extracts all relevant MS1 and MS2 data, respectively, then discards data outside of the mass range(s) calculated from the provided mz and ppm. The default, "everything", includes all MS1, MS2, BPC, TIC, and metadata.

`verbosity` Three levels of processing output to the R console are available, with increasing verbosity corresponding to higher integers. A verbosity of zero means that no output will be produced, useful when wrapping within larger functions. A verbosity of 1 will produce a progress bar using base R's txtProgressBar function. A verbosity of 2 or higher will produce timing output for each individual file read in. The default, NULL, will select between 1 and 2 depending on the number of files being read: if a single file, verbosity is set to 2; if multiple files, verbosity is set to 1.

`incl_polarity` Toggle this option to TRUE for mixed-polarity files. An additional column will be added corresponding to the polarity of the scan, with either a 1 or a -1 corresponding to positive and negative mode, respectively.

`mz` A vector of the mass-to-charge ratio for compounds of interest. Only used when combined with 'grab_what = "EIC"' (see above). Multiple masses can be provided.

`ppm` A single number corresponding to the mass accuracy (in parts per million) of the instrument on which the data was collected. Only used when combined with 'grab_what = "EIC"' (see above).

`rtrange` Only available when parsing mzML files. A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object's size.

`prefilter` A single number corresponding to the minimum intensity of interest in the MS1 data. Data points with intensities below this threshold will be silently dropped, which can dramatically reduce the size of the final object. Currently only works with MS1 data, but could be expanded easily to handle more.

Value

A list of 'data.table's, each named after the arguments requested in grab_what. E.g. $MS1 contains MS1 information, $MS2 contains fragmentation info, etc. MS1 data has four columns: retention time (rt), mass-to-charge (mz), intensity (int), and filename. MS2 data has six: retention time (rt), precursor m/z (premz), fragment m/z (fragmz), fragment intensity (int), collision energy (voltage), and filename. MS3 adds an additional column to this (prepremz) corresponding to the initial MS1 m/z targeted. Data requested that does not exist in the provided files (such as MS2 data requested from MS1-only files) will return an empty (length zero) data.table. The data.tables extracted from each of the individual files are collected into one large table using data.table's 'rbindlist'. $metadata is a little weirder because the metadata doesn't fit neatly into a tidy format but things are hopefully named helpfully. $chroms was added in v1.3 and contains 7 columns: chromatogram type (usually TIC, BPC or SRM info), chromatogram index, target mz, product mz, retention time (rt), and intensity (int). $DAD was also added in v1.3 and contains has three columns: retention time (rt), wavelength (lambda),and intensity (int). Data requested that does not exist in the provided files (such as MS2 data requested from MS1-only files) will return an empty (zero-row) data.table.

Examples

```
library(RaMS)

# Extract MS1 data from a couple files
```

```
sample_dir <- system.file("extdata", package = "RaMS")
sample_files <- list.files(sample_dir, full.names=TRUE)
multifile_data <- grabMSdata(sample_files[c(3,5,6)], grab_what="MS1")
```

---

grabMzmlBPC *Grab the BPC or TIC from a file*

Description

The base peak intensity and total ion current are actually written into the mzML files and aren't encoded, making retrieval of BPC and TIC information blazingly fast if parsed correctly.

Usage

```
grabMzmlBPC(xml_data, rtrange, TIC = FALSE, incl_polarity)
```

Arguments

xml_data	An 'xml2' nodeset, usually created by applying 'read_xml' to an mzML file.
rtrange	A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object's size.
TIC	Boolean. If TRUE, the TIC is extracted rather than the BPC.
incl_polarity	Boolean determining whether the polarity of the scan should be returned as a column in the table (positive mode = 1, negative mode = -1)

Value

A 'data.table' with columns for retention time (rt), and intensity (int).

---

grabMzmlDAD *Extract the DAD data from an mzML nodeset*

Description

Extract the DAD data from an mzML nodeset

Usage

```
grabMzmlDAD(xml_data, rtrange, file_metadata)
```

Arguments

xml_data	An ‘xml2‘ nodeset, usually created by applying ‘read_xml‘ to an mzML file.
rtrange	A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object’s size.
file_metadata	Information about the file used to decode the binary arrays containing m/z and intensity information.

Value

A ‘data.table‘ with columns for retention time (rt), wavelength (lambda), and intensity (int).

---

grabMzmlData *Get mass-spectrometry data from an mzML file*

Description

This function handles the mzML side of things, reading in files that are written in the mzML format. Much of the code is similar to the mzXML format, but the xpath handles are different and the mz/int array is encoded as two separate entries rather than simultaneously. This function has been exposed to the user in case per-file optimization (such as peakpicking or additional filtering) is desired before the full data object is returned.

Usage

```
grabMzmlData(
  filename,
  grab_what,
  verbosity = 0,
  incl_polarity = FALSE,
  mz = NULL,
  ppm = NULL,
  rtrange = NULL,
  prefilter = -1
)
```

Arguments

filename	A single filename to read into R’s memory. Both absolute and relative paths are acceptable.
grab_what	What data should be read from the file? Options include "MS1" for data only from the first spectrometer, "MS2" and "MS3" for fragmentation data, "BPC" for rapid access to the base peak chromatogram, "TIC" for rapid access to the total ion chromatogram, "DAD" for DAD (UV) data, and "chroms" for precompiled chromatogram data (especially useful for MRM but often contains BPC/TIC in

other files). Metadata can be accessed with "metadata", which provides information about the instrument and time the file was run. These options can be combined (i.e. ‘grab_data=c("MS1", "MS2", "BPC")‘) or this argument can be set to "everything" to extract all of the above. Options "EIC", "EIC_MS2", and "EIC_MS3" are useful when working with files whose total size exceeds working memory - it first extracts all relevant MS1/2/3 data, respectively, then discards data outside of the mass range(s) calculated from the provided mz and ppm. The default, "everything", includes all MS1, MS2, BPC, TIC, and metadata.

`verbosity` Three levels of processing output to the R console are available, with increasing verbosity corresponding to higher integers. A verbosity of zero means that no output will be produced, useful when wrapping within larger functions. A verbosity of 1 will produce a progress bar using base R's txtProgressBar function. A verbosity of 2 or higher will produce timing output for each individual file read in.

`incl_polarity` Toggle this option to TRUE for mixed-polarity files. An additional column will be added corresponding to the polarity of the scan, with either a 1 or a -1 corresponding to positive and negative mode, respectively.

`mz` A vector of the mass-to-charge ratio for compounds of interest. Only used when combined with ‘grab_what = "EIC"‘ (see above). Multiple masses can be provided.

`ppm` A single number corresponding to the mass accuracy (in parts per million) of the instrument on which the data was collected. Only used when combined with ‘grab_what = "EIC"‘ (see above).

`rtrange` A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object's size.

`prefilter` A single number corresponding to the minimum intensity of interest in the MS1 data. Data points with intensities below this threshold will be silently dropped, which can dramatically reduce the size of the final object. Currently only works with MS1 data, but could be expanded easily to handle more.

Value

A list of ‘data.table‘s, each named after the arguments requested in grab_what. E.g. $MS1 contains MS1 information, $MS2 contains fragmentation info, etc. MS1 data has four columns: retention time (rt), mass-to-charge (mz), intensity (int), and filename. MS2 data has six: retention time (rt), precursor m/z (premz), fragment m/z (fragmz), fragment intensity (int), collision energy (voltage), and filename. MS3 has an additional column to MS2 (prepremz) which has the original MS1 scan's m/z ratio. Data requested that does not exist in the provided files (such as MS2 data requested from MS1-only files) will return an empty (length zero) data.table. The data.tables extracted from each of the individual files are collected into one large table using data.table's ‘rbindlist‘. $metadata is a little weirder because the metadata doesn't fit neatly into a tidy format but things are hopefully named helpfully. $chroms was added in v1.3 and contains 7 columns: chromatogram type (usually TIC, BPC or SRM info), chromatogram index, target mz, product mz, retention time (rt), and intensity (int). $DAD was also added in v1.3 and contains has three columns: retention time (rt), wavelength (lambda),and intensity (int).

Examples

```
sample_file <- system.file("extdata", "LB12HL_AB.mzML.gz", package = "RaMS")
file_data <- grabMzmlData(sample_file, grab_what="MS1")
## Not run:
# Extract MS1 data and a base peak chromatogram
file_data <- grabMzmlData(sample_file, grab_what=c("MS1", "BPC"))
# Extract data from a retention time subset
file_data <- grabMzmlData(sample_file, grab_what=c("MS1", "BPC"),
                          rtrange=c(5, 7))
# Extract EIC for a specific mass
file_data <- grabMzmlData(sample_file, grab_what="EIC", mz=118.0865, ppm=5)
# Extract EIC for several masses simultaneously
file_data <- grabMzmlData(sample_file, grab_what="EIC", ppm=5,
                          mz=c(118.0865, 146.118104, 189.123918))

# Extract MS2 data
sample_file <- system.file("extdata", "S30657.mzML.gz", package = "RaMS")
MS2_data <- grabMzmlData(sample_file, grab_what="MS2")

## End(Not run)
```

---

grabMzmlEncodingData *Helper function to extract mzML file encoding data*

Description

Helper function to extract mzML file encoding data

Usage

```
grabMzmlEncodingData(xml_data)
```

Arguments

xml_data	mzML data as parsed by xml2

Value

A list of values used by other parsing functions, currently compression, mz_precision, int_precision

grabMzmlMetadata *Helper function to extract mzML file metadata*

Description

Helper function to extract mzML file metadata

Usage

```
grabMzmlMetadata(xml_data)
```

Arguments

xml_data	mzML data as parsed by xml2

Value

A list of values corresponding to various pieces of metadata for each file

grabMzmlMS1 *Extract the MS1 data from an mzML nodeset*

Description

Extract the MS1 data from an mzML nodeset

Usage

```
grabMzmlMS1(xml_data, rtrange, file_metadata, prefilter, incl_polarity)
```

Arguments

xml_data	An ‘xml2‘ nodeset, usually created by applying ‘read_xml‘ to an mzML file.
rtrange	A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object’s size.
file_metadata	Information about the file used to decode the binary arrays containing m/z and intensity information.
prefilter	The lowest intensity value of interest, used to reduce file size (and especially useful for profile mode data with many 0 values)
incl_polarity	Boolean determining whether the polarity of the scan should be returned as a column in the table (positive mode = 1, negative mode = -1)

Value

A ‘data.table‘ with columns for retention time (rt), m/z (mz), and intensity (int).

grabMzmlMS2 *Extract the MS2 data from an mzML nodeset*

Description

Extract the MS2 data from an mzML nodeset

Usage

```
grabMzmlMS2(xml_data, rtrange, file_metadata, incl_polarity)
```

Arguments

xml_data	An ‘xml2‘ nodeset, usually created by applying ‘read_xml‘ to an mzML file.
rtrange	A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object’s size.
file_metadata	Information about the file used to decode the binary arrays containing m/z and intensity information.
incl_polarity	Boolean determining whether the polarity of the scan should be returned as a column in the table (positive mode = 1, negative mode = -1)

Value

A ‘data.table‘ with columns for retention time (rt), precursor m/z (premz), fragment m/z (fragmz), collision energy (voltage), and intensity (int).

grabMzmlMS3 *Extract the MS3 data from an mzML nodeset*

Description

Extract the MS3 data from an mzML nodeset

Usage

```
grabMzmlMS3(xml_data, rtrange, file_metadata, incl_polarity)
```

Arguments

xml_data	An ‘xml2‘ nodeset, usually created by applying ‘read_xml‘ to an mzML file.
rtrange	A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object’s size.
file_metadata	Information about the file used to decode the binary arrays containing m/z and intensity information.
incl_polarity	Boolean determining whether the polarity of the scan should be returned as a column in the table (positive mode = 1, negative mode = -1)

Value

A ‘data.table‘ with columns for retention time (rt), MS1 precursor m/z (prepremz), MS2 precursor m/z (premz), fragment m/z (fragmz), collision energy (voltage), and intensity (int).

---

grabMzxmlBPC *Grab the BPC or TIC from a file*

Description

The base peak intensity and total ion current are actually written into the mzXML files and aren’t encoded, making retrieval of BPC and TIC information blazingly fast if parsed correctly.

Usage

```
grabMzxmlBPC(xml_data, TIC = FALSE, rtrange, incl_polarity)
```

Arguments

xml_data	An ‘xml2‘ nodeset, usually created by applying ‘read_xml‘ to an mzML file.
TIC	Boolean. If TRUE, the TIC is extracted rather than the BPC.
rtrange	A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object’s size.
incl_polarity	Boolean determining whether the polarity of the scan should be returned as a column in the table (positive mode = 1, negative mode = -1)

Value

A ‘data.table‘ with columns for retention time (rt), and intensity (int).

grabMzxmlData *Get mass-spectrometry data from an mzXML file*

Description

This function handles the mzXML side of things, reading in files that are written in the mzXML format. Much of the code is similar to the mzXML format, but the xpath handles are different and the mz/int array is encoded simultaneously rather than as two separate entries. This function has been exposed to the user in case per-file optimization (such as peakpicking or additional filtering) is desired before the full data object is returned.

Usage

```
grabMzxmlData(
  filename,
  grab_what,
  verbosity = 0,
  incl_polarity = FALSE,
  rtrange = NULL,
  mz = NULL,
  ppm = NULL,
  prefilter = -1
)
```

Arguments

filename	A single filename to read into R's memory. Both absolute and relative paths are acceptable.
grab_what	What data should be read from the file? Options include "MS1" for data only from the first spectrometer, "MS2" for fragmentation data, "BPC" for rapid access to the base peak chromatogram, and "TIC" for rapid access to the total ion chromatogram. DAD and chromatogram ("DAD" and "chroms") are unavailable for mzXML files. Metadata can be accessed with "metadata", which provides information about the instrument and time the file was run. These options can be combined (i.e. 'grab_data=c("MS1", "MS2", "BPC")') or this argument can be set to "everything" to extract all of the above. Options "EIC" and "EIC_MS2" are useful when working with files whose total size exceeds working memory - it first extracts all relevant MS1 and MS2 data, respectively, then discards data outside of the mass range(s) calculated from the provided mz and ppm. The default, "everything", includes all MS1, MS2, BPC, TIC, and metadata.
verbosity	Three levels of processing output to the R console are available, with increasing verbosity corresponding to higher integers. A verbosity of zero means that no output will be produced, useful when wrapping within larger functions. A verbosity of 1 will produce a progress bar using base R's txtProgressBar function. A verbosity of 2 or higher will produce timing output for each individual file read in.

`incl_polarity` Toggle this option to TRUE for mixed-polarity files. An additional column will be added corresponding to the polarity of the scan, with either a 1 or a -1 corresponding to positive and negative mode, respectively.

`rtrange` Not supported for mzXML data. Only provided here so as to throw a friendly warning rather than an unexpected error.

`mz` A vector of the mass-to-charge ratio for compounds of interest. Only used when combined with ‘grab_what = "EIC"‘ (see above). Multiple masses can be provided.

`ppm` A single number corresponding to the mass accuracy (in parts per million) of the instrument on which the data was collected. Only used when combined with ‘grab_what = "EIC"‘ (see above).

`prefilter` A single number corresponding to the minimum intensity of interest in the MS1 data. Data points with intensities below this threshold will be silently dropped, which can dramatically reduce the size of the final object. Currently only works with MS1 data, but could be expanded easily to handle more.

Value

A list of ‘data.table‘s, each named after the arguments requested in grab_what. E.g. $MS1 contains MS1 information, $MS2 contains fragmentation info, etc. MS1 data has four columns: retention time (rt), mass-to-charge (mz), intensity (int), and filename. MS2 data has six: retention time (rt), precursor m/z (premz), fragment m/z (fragmz), fragment intensity (int), collision energy (voltage), and filename. MS3 has an additional column to MS2 (prepremz) which has the original MS1 scan's m/z ratio. Data requested that does not exist in the provided files (such as MS2 data requested from MS1-only files) will return an empty (length zero) data.table. The data.tables extracted from each of the individual files are collected into one large table using data.table's ‘rbindlist‘. $metadata is a little weirder because the metadata doesn't fit neatly into a tidy format but things are hopefully named helpfully.

Examples

```
sample_file <- system.file("extdata", "LB12HL_AB.mzXML.gz", package = "RaMS")
file_data <- grabMzxmlData(sample_file, grab_what="MS1")
## Not run:
# Extract MS1 data and a base peak chromatogram
file_data <- grabMzxmlData(sample_file, grab_what=c("MS1", "BPC"))
# Extract EIC for a specific mass
file_data <- grabMzxmlData(sample_file, grab_what="EIC", mz=118.0865, ppm=5)
# Extract EIC for several masses simultaneously
file_data <- grabMzxmlData(sample_file, grab_what="EIC", ppm=5,
                           mz=c(118.0865, 146.118104, 189.123918))

# Extract MS2 data
sample_file <- system.file("extdata", "S30657.mzXML.gz", package = "RaMS")
MS2_data <- grabMzxmlData(sample_file, grab_what="MS2")

## End(Not run)
```

grabMzxmlEncodingData *Helper function to extract mzXML file metadata*

Description

Helper function to extract mzXML file metadata

Usage

```
grabMzxmlEncodingData(xml_data)
```

Arguments

xml_data	mzXML data as parsed by xml2

Value

A list of values used by other parsing functions, currently compression, precision, and endian encoding (endi_enc)

grabMzxmlMetadata *Helper function to extract mzXML file metadata*

Description

Helper function to extract mzXML file metadata

Usage

```
grabMzxmlMetadata(xml_data)
```

Arguments

xml_data	mzXML data as parsed by xml2

Value

A list of values corresponding to various pieces of metadata for each file

grabMzxmlMS1 *Extract the MS1 data from an mzXML nodeset*

Description

Extract the MS1 data from an mzXML nodeset

Usage

```
grabMzxmlMS1(xml_data, file_metadata, rtrange, prefilter, incl_polarity)
```

Arguments

xml_data	An ‘xml2‘ nodeset, usually created by applying ‘read_xml‘ to an mzXML file.
file_metadata	Information about the file used to decode the binary arrays containing m/z and intensity information.
rtrange	A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object’s size.
prefilter	The lowest intensity value of interest, used to reduce file size (and especially useful for profile mode data with many 0 values)
incl_polarity	Boolean determining whether the polarity of the scan should be returned as a column in the table (positive mode = 1, negative mode = -1)

Value

A ‘data.table‘ with columns for retention time (rt), m/z (mz), and intensity (int).

grabMzxmlMS2 *Extract the MS2 data from an mzXML nodeset*

Description

Extract the MS2 data from an mzXML nodeset

Usage

```
grabMzxmlMS2(xml_data, file_metadata, rtrange, incl_polarity)
```

Arguments

xml_data	An ‘xml2‘ nodeset, usually created by applying ‘read_xml‘ to an mzXML file.
file_metadata	Information about the file used to decode the binary arrays containing m/z and intensity information.
rtrange	A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object's size.
incl_polarity	Boolean determining whether the polarity of the scan should be returned as a column in the table (positive mode = 1, negative mode = -1)

Value

A ‘data.table‘ with columns for retention time (rt), precursor m/z (premz), fragment m/z (fragmz), collision energy (voltage), and intensity (int).

---

grabMzxmlMS3 *Extract the MS3 data from an mzXML nodeset*

Description

Extract the MS3 data from an mzXML nodeset

Usage

```
grabMzxmlMS3(xml_data, file_metadata, rtrange, incl_polarity)
```

Arguments

xml_data	An ‘xml2‘ nodeset, usually created by applying ‘read_xml‘ to an mzXML file.
file_metadata	Information about the file used to decode the binary arrays containing m/z and intensity information.
rtrange	A vector of length 2 containing an upper and lower bound on retention times of interest. Providing a range here can speed up load times (although not enormously, as the entire file must still be read) and reduce the final object's size.
incl_polarity	Boolean determining whether the polarity of the scan should be returned as a column in the table (positive mode = 1, negative mode = -1)

Value

A ‘data.table‘ with columns for retention time (rt), MS1 precursor m/z (prepremz), MS2 precursor m/z (premz), fragment m/z (fragmz), collision energy (voltage), and intensity (int).

grabMzxmlSpectraMzInt *Extract the mass-to-charge data from the spectra of an mzXML nodeset*

Description

The mz and intensity information of mzXML files are encoded as a binary array, sometimes compressed via gzip or zlib or numpress. This code finds all the m/z-int binary arrays and converts them back to the original measurements. See https://github.com/ProteoWizard/pwiz/issues/1301

Usage

```
grabMzxmlSpectraMzInt(xml_nodes, file_metadata)
```

Arguments

xml_nodes	An xml_nodeset object corresponding to the spectra collected by the mass spectrometer, usually produced by applying 'xml_find_all' to an MS1 or MS2 nodeset.
file_metadata	Information about the file used to decode the binary arrays containing m/z and intensity information. Here, the compression and mz precision information is relevant.

Value

A numeric vector of masses, many for each scan.

grabMzxmlSpectraPremz *Extract the precursor mass from the spectra of an mzXML nodeset*

Description

Extract the precursor mass from the spectra of an mzXML nodeset

Usage

```
grabMzxmlSpectraPremz(xml_nodes)
```

Arguments

xml_nodes	An xml_nodeset object corresponding to the spectra collected by the mass spectrometer, usually produced by applying 'xml_find_all' to an MS1 or MS2 nodeset.

Value

A numeric vector of precursor masses, one for each scan

grabMzxmlSpectraRt *Extract the retention time from the spectra of an mzXML nodeset*

Description

Extract the retention time from the spectra of an mzXML nodeset

Usage

```
grabMzxmlSpectraRt(xml_nodes)
```

Arguments

xml_nodes	An xml_nodeset object corresponding to the spectra collected by the mass spectrometer, usually produced by applying ‘xml_find_all‘ to an MS1 or MS2 nodeset.

Value

A numeric vector of retention times, one for each scan

grabMzxmlSpectraVoltage *Extract the collision energies from the spectra of an mzXML nodeset*

Description

Although the collision energy is typically fixed per file, it's equally fast (afaik) to just grab them all individually here. Also, I'm worried about these rumors of "ramped" collision energies

Usage

```
grabMzxmlSpectraVoltage(xml_nodes)
```

Arguments

xml_nodes	An xml_nodeset object corresponding to the spectra collected by the mass spectrometer, usually produced by applying ‘xml_find_all‘ to an MS1 or MS2 nodeset.

Value

A numeric vector of collision energies, one for each scan.

grabSpectraInt *Extract the intensity information from the spectra of an mzML nodeset*

Description

The mz and intensity information of mzML files are encoded as binary arrays, sometimes compressed via gzip or zlib or numpress. This code finds all the intensity binary arrays and converts them back to the original measurements. See https://github.com/ProteoWizard/pwiz/issues/1301

Usage

```
grabSpectraInt(xml_nodes, file_metadata)
```

Arguments

xml_nodes	An xml_nodeset object corresponding to the spectra collected by the mass spectrometer, usually produced by applying 'xml_find_all' to an MS1 or MS2 nodeset.
file_metadata	Information about the file used to decode the binary arrays containing m/z and intensity information. Here, the compression and int precision information is relevant.

Value

A numeric vector of intensities, many for each scan.

grabSpectraMz *Extract the mass-to-charge data from the spectra of an mzML nodeset*

Description

The mz and intensity information of mzML files are encoded as binary arrays, sometimes compressed via gzip or zlib or numpress. This code finds all the m/z binary arrays and converts them back to the original measurements. See https://github.com/ProteoWizard/pwiz/issues/1301

Usage

```
grabSpectraMz(xml_nodes, file_metadata)
```

Arguments

xml_nodes	An xml_nodeset object corresponding to the spectra collected by the mass spectrometer, usually produced by applying 'xml_find_all' to an MS1 or MS2 nodeset.
file_metadata	Information about the file used to decode the binary arrays containing m/z and intensity information. Here, the compression and mz precision information is relevant.

Value

A numeric vector of masses, many for each scan.

`grabSpectraPremz` *Extract the precursor mass from the spectra of an mzML nodeset*

Description

Extract the precursor mass from the spectra of an mzML nodeset

Usage

```
grabSpectraPremz(xml_nodes)
```

Arguments

`xml_nodes`	An xml_nodeset object corresponding to the spectra collected by the mass spectrometer, usually produced by applying 'xml_find_all' to an MS1 or MS2 nodeset.

Value

A numeric vector of precursor masses, one for each scan

`grabSpectraRt` *Extract the retention time from the spectra of an mzML nodeset*

Description

Extract the retention time from the spectra of an mzML nodeset

Usage

```
grabSpectraRt(xml_nodes)
```

Arguments

`xml_nodes`	An xml_nodeset object corresponding to the spectra collected by the mass spectrometer, usually produced by applying 'xml_find_all' to an MS1 or MS2 nodeset.

Value

A numeric vector of retention times, one for each scan

grabSpectraVoltage *Extract the collision energies from the spectra of an mzML nodeset*

Description

Although the collision energy is typically fixed per file, it's equally fast (afaik) to just grab them all individually here. Also, I'm worried about these rumors of "ramped" collision energies

Usage

```
grabSpectraVoltage(xml_nodes)
```

Arguments

xml_nodes	An xml_nodeset object corresponding to the spectra collected by the mass spectrometer, usually produced by applying 'xml_find_all' to an MS1 or MS2 nodeset.

Value

A numeric vector of collision energies, one for each scan.

minifyMSdata *Shrink MS data by including only data points near masses of interest*

Description

MS files can be annoyingly large if only a few masses are of interest. This large size makes it difficult to share them online for debugging purposes and often means that untargeted algorithms spend a lot of time picking peaks in data that's irrelevant. minifyMSdata is a function designed to "minify" MS files by extracting only those data points that are within the ppm error of an m/z value of interest, and returns the file essentially otherwise unchanged.

Usage

```
minifyMSdata(
  files,
  output_files = NULL,
  mz_exclude = NULL,
  mz_include = NULL,
  ppm = NULL,
  warn = TRUE,
  prefilter = -1,
  verbosity = NULL
)
```

Arguments

files	The name of a single file to be minified, usually produced by Proteowizard's 'msconvert' or something similar.
output_files	The name of the file to be written out.
mz_exclude	A vector of m/z values that should be excluded from the minified file. This argument must be used with the 'ppm' argument and should not be used with mz_include. For each mass provided, an m/z window of +/- 'ppm' is calculated, and all data points within that window are removed.
mz_include	A vector of m/z values that should be included in the minified file. This argument must be used with the 'ppm' argument and should not be used with mz_exclude. For each mass provided, an m/z window of +/- 'ppm' is calculated, and all data points within that window are kept.
ppm	The parts-per-million error of the instrument used to collect the original file.
warn	Boolean. Should the function warn the user when removing an index from an mzML file?
prefilter	A single number corresponding to the minimum intensity of interest in the MS1 data. Data points with intensities below this threshold will be silently dropped, which can dramatically reduce the size of the final object. Currently only works with MS1 data, but could be expanded easily to handle more.
verbosity	A single number with a sensible default behavior. If larger than 2, will render a progress bar as files are processed.

Value

Invisibly, the name of the new files.

Examples

```
## Not run:
library(RaMS)
# Extract data corresponding to only valine and homarine
# m/z = 118.0865 and 138.0555, respectively
filename <- system.file("extdata", "LB12HL_AB.mzML.gz", package = "RaMS")
output_filename <- "mini_LB12HL_AB.mzML"
include_mzs <- c(118.0865, 138.0555)
minifyMSdata(filename, output_filename, mz_include=include_mzs, ppm=5)
init_data <- grabMSdata(filename)
mini_data <- grabMSdata(output_filename)
qplotMS1data(rbind(init_data$BPC, mini_data$BPC), color_col = "filename")
unlink(output_filename)

# Exclude data corresponding to valine and homarine
filename <- system.file("extdata", "LB12HL_AB.mzML.gz", package = "RaMS")
output_filename <- "mini_LB12HL_AB.mzML"
exclude_mzs <- c(118.0865, 138.0555)
minifyMSdata(filename, output_filename, mz_exclude=exclude_mzs, ppm=5)
init_data <- grabMSdata(filename)
mini_data <- grabMSdata(output_filename)
```

```
qplotMS1data(rbind(init_data$BPC, mini_data$BPC), color_col = "filename")
unlink(output_filename)

## End(Not run)
```

`minifyMzml` *Shrink mzML files by including only data points near masses of interest*

Description

mzML files can be annoyingly large if only a few masses are of interest. This large size makes it difficult to share them online for debugging purposes and often means that untargeted algorithms spend a lot of time picking peaks in data that's irrelevant. minifyMzml is a function designed to "minify" mzML files by extracting only those data points that are within a ppm error of an m/z value of interest, and returns the file essentially otherwise unchanged. This function currently works only on MS1 data, but is reasonably expandable if demand becomes evident.

Usage

```
minifyMzml(
  filename,
  output_filename,
  ppm,
  mz_exclude = NULL,
  mz_include = NULL,
  warn = TRUE,
  prefilter = -1
)
```

Arguments

`filename`	The name of a single file to be minified, usually produced by Proteowizard's 'msconvert' or something similar.
`output_filename`	The name of the file to be written out.
`ppm`	The parts-per-million error of the instrument used to collect the original file.
`mz_exclude`	A vector of m/z values that should be excluded from the minified file. This argument must be used with the 'ppm' argument and should not be used with mz_include. For each mass provided, an m/z window of +/- 'ppm' is calculated, and all data points within that window are removed.
`mz_include`	A vector of m/z values that should be included in the minified file. This argument must be used with the 'ppm' argument and should not be used with mz_exclude. For each mass provided, an m/z window of +/- 'ppm' is calculated, and all data points within that window are kept.
`warn`	Boolean. Should the function warn the user when removing an index from an mzML file?

prefilter — A single number corresponding to the minimum intensity of interest in the MS1 data. Data points with intensities below this threshold will be silently dropped, which can dramatically reduce the size of the final object. Currently only works with MS1 data, but could be expanded easily to handle more.

Value

Invisibly, the name of the new file.

Examples

```
## Not run:
library(RaMS)
# Extract data corresponding to only valine and homarine
# m/z = 118.0865 and 138.0555, respectively
filename <- system.file("extdata", "LB12HL_AB.mzML.gz", package = "RaMS")
output_filename <- "mini_LB12HL_AB.mzML"
include_mzs <- c(118.0865, 138.0555)
minifyMzml(filename, output_filename, mz_include=include_mzs, ppm=5)
unlink(output_filename)

# Exclude data corresponding to valine and homarine
filename <- system.file("extdata", "LB12HL_AB.mzML.gz", package = "RaMS")
output_filename <- "mini_LB12HL_AB.mzML"
exclude_mzs <- c(118.0865, 138.0555)
minifyMzml(filename, output_filename, mz_exclude=exclude_mzs, ppm=5)
unlink(output_filename)

## End(Not run)
```

---

minifyMzxml — *Shrink mzXML files by including only data points near masses of interest*

Description

mzXML files can be annoyingly large if only a few masses are of interest. This large size makes it difficult to share them online for debugging purposes and often means that untargeted algorithms spend a lot of time picking peaks in data that's irrelevant. minifyMzxml is a function designed to "minify" mzXML files by extracting only those data points that are within a ppm error of an m/z value of interest, and returns the file essentially otherwise unchanged. This function currently works only on MS1 data, but is reasonably expandable if demand becomes evident.

Usage

```
minifyMzxml(
  filename,
  output_filename,
  ppm,
```

```
  mz_exclude = NULL,
  mz_include = NULL,
  prefilter = -1,
  warn = TRUE
)
```

Arguments

filename	The name of a single file to be minified, usually produced by Proteowizard's 'msconvert' or something similar.
output_filename	The name of the file to be written out.
ppm	The parts-per-million error of the instrument used to collect the original file.
mz_exclude	A vector of m/z values that should be excluded from the minified file. This argument must be used with the 'ppm' argument and should not be used with mz_include. For each mass provided, an m/z window of +/- 'ppm' is calculated, and all data points within that window are removed.
mz_include	A vector of m/z values that should be included in the minified file. This argument must be used with the 'ppm' argument and should not be used with mz_exclude. For each mass provided, an m/z window of +/- 'ppm' is calculated, and all data points within that window are kept.
prefilter	A single number corresponding to the minimum intensity of interest in the MS1 data. Data points with intensities below this threshold will be silently dropped, which can dramatically reduce the size of the final object. Currently only works with MS1 data, but could be expanded easily to handle more.
warn	Boolean. Should the function warn the user when removing an index from an mzML file?

Value

Invisibly, the name of the new file.

Examples

```
## Not run:
library(RaMS)
# Extract data corresponding to only valine and homarine
# m/z = 118.0865 and 138.0555, respectively
filename <- system.file("extdata", "LB12HL_AB.mzXML.gz", package = "RaMS")
output_filename <- "mini_LB12HL_AB.mzXML"
include_mzs <- c(118.0865, 138.0555)
minifyMzxml(filename, output_filename, mz_include=include_mzs, ppm=5)
unlink(output_filename)

# Exclude data corresponding to valine and homarine
filename <- system.file("extdata", "LB12HL_AB.mzXML.gz", package = "RaMS")
output_filename <- "mini_LB12HL_AB.mzXML"
exclude_mzs <- c(118.0865, 138.0555)
minifyMzxml(filename, output_filename, mz_exclude=exclude_mzs, ppm=5)
```

```
unlink(output_filename)

## End(Not run)
```

`msdata_connection` *S3 constructor for msdata_connection*

Description

S3 constructor for msdata_connection

Usage

```
msdata_connection(x)
```

Arguments

`x`	This is a thing?

Value

Itself, with the class?

`mz_group` *Group m/z values into bins of a specified ppm width*

Description

This function bins m/z values based on their proximity to each other in m/z space. The algorithm takes the first value in the m/z vector and uses that as the center of a window with a ppm value provided by the user and assigns all m/z values within that window to the same group, then removes those values from consideration and repeats the process until there are no points left to group. This is often used to construct chromatograms from raw MS data that can then be visualized or peakpicked. The function can also drop groups of m/z values if there's not enough points within them or produce only a certain number of groups. Because the algorithm uses the first value in the m/z vector as the window center, it's often a good idea to first sort the values by decreasing intensity.

Usage

```
mz_group(mz_vals, ppm, min_group_size = 0, max_groups = NULL)
```

Arguments

mz_vals	A numeric vector of m/z values
ppm	A length-1 numeric vector specifying the desired window size in ppm
min_group_size	A length-1 numeric vector specifying the minimum number of points that must fall within an m/z window to be assigned a group number
max_groups	A length-1 numeric vector specifying the maximum number of total groups to assign.

Value

A numeric vector of the same length as mz_vals specifying the group into which each m/z value was binned. Values not assigned to a group are returned as NAs.

Examples

```
example_mz_vals <- c(118.0, 118.1, 138.0, 152.0, 118.2, 138.1, 118.1)
mz_group(example_mz_vals, ppm = 1)
mz_group(example_mz_vals, ppm = 1000)
mz_group(example_mz_vals, ppm = 200000)

mz_group(example_mz_vals, ppm = 1000, min_group_size = 2)
mz_group(example_mz_vals, ppm = 1000, max_groups = 2)

## Not run:
sample_dir <- system.file("extdata", package = "RaMS")
sample_files <- list.files(sample_dir, full.names=TRUE)
msdata <- grabMSdata(sample_files[c(3, 5, 6)], grab_what="MS1")

grouped_MS1 <- msdata$MS1[mz%between%pmppm(119.0865, 100)][
 order(int, decreasing = TRUE)][
   ,mz_group:=mz_group(mz, ppm = 5)][]
print(grouped_MS1)

library(ggplot2)
library(dplyr)
msdata$MS1[mz%between%pmppm(119.0865, 100)] %>%
  arrange(desc(int)) %>%
  mutate(mz_group=mz_group(mz, ppm=10)) %>%
  ggplot() +
  geom_point(aes(x=rt, y=mz, color=factor(mz_group)))

msdata$MS1[mz%between%pmppm(119.0865, 100)] %>%
  arrange(desc(int)) %>%
  mutate(mz_group=mz_group(mz, ppm=5)) %>%
  qplotMS1data(facet_col = "mz_group")
msdata$MS1[mz%between%pmppm(119.0865, 100)] %>%
  arrange(desc(int)) %>%
  mutate(mz_group=mz_group(mz, ppm=5, max_groups = 2)) %>%
  qplotMS1data(facet_col = "mz_group")

## End(Not run)
```

node2dt *Convert node to data.table*

Description

Convert node to data.table

Usage

```
node2dt(dubset_node, ms_level)
```

Arguments

dubset_node	The "data subset" node with children rt, mz, etc.
ms_level	The requested MS level to search for

Value

A data.table with columns depending on the MS level requested

pmppm *Plus/minus parts per million*

Description

It shouldn't be hard to translate a point mass into a mass window bounded by spectrometer accuracy.

Usage

```
pmppm(mass, ppm = 4)
```

Arguments

mass	A length-1 numeric representing the mass of interest for which a mass range is desired.
ppm	The parts-per-million accuracy of the mass spectrometer on which the data was collected.

Value

A length-2 numeric representing the mass range requested

Examples

```
pmppm(100, 5)
pmppm(1000000, 5)
pmppm(118.0865, 2.5)
pmppm(892.535313, 10)
```

print.msdata_connection *S3 print option for msdata_connection objects*

Description

S3 print option for msdata_connection objects

Usage

```
## S3 method for class 'msdata_connection'
print(x, ...)
```

Arguments

x	An msdata_connection object containing files and grab_what
...	Other arguments to be passed to print.default, I guess

Value

Messages, mostly

qplotMS1data *Quick plot for MS data*

Description

Syntactic sugar for a common chromatogram plot. Will use ‘ggplot2‘ if available but has a base plot implementation for use even in ultra lightweight situations. Accepts the default MS1 output from ‘grabMSdata‘ of a data.table (or base data.frame) with columns for rt (retention time) and int (intensity) as well as filename. Creates a plot of intensity vs retention time with one trace per file. A few additional ‘ggplot2‘ arguments are also made available for easy coloring or facetting by providing the name of the associated column to the ‘color_col‘ and ‘facet_col‘ arguments, respectively.

Usage

```
qplotMS1data(
  MS1_df,
  color_col = NULL,
  facet_col = NULL,
  facet_args = list(ncol = 1),
  force_base = FALSE
)
```

Arguments

MS1_df	A data.table with at least three columns named rt, int, and filename
color_col	The name of the column to color by. Must be quoted.
facet_col	The name of the column to facet by. Must be quoted.
facet_args	Since the call to facet_wrap is within the function, you can provide additional facet customization arguments here as a list. Although if you're starting to fiddle with facets you'll probably be better served by the proper 'ggplot' call.
force_base	Boolean option to force base R graphics instead of 'ggplot' even if the 'ggplot2' package is installed.

Value

If 'ggplot2' is installed, a 'ggplot' object that can be further modified via additional + commands. Otherwise, NULL and the plot appears via base graphics at the active device.

Examples

```
test_df <- expand.grid(rt=rep(1:100, length.out=1000))
test_df$int <- rep(dnorm(seq(-10, 10, length.out=100)), 10)*10+runif(1000)
test_df$filename <- rep(LETTERS[1:10], each=100)
qplotMS1data(test_df)

test_df$startime <- rep(gl(2, 5, labels = c("Morn", "Eve")), each=100)
qplotMS1data(test_df, color_col="startime", facet_col="startime")
qplotMS1data(test_df, color_col="startime", facet_col="startime",
            facet_args=list(ncol=2, scales="free"))

# Using data from the `grabMSdata` function:
## Not run:
sample_dir <- system.file("extdata", package = "RaMS")
sample_files <- list.files(sample_dir, full.names=TRUE)
msdata <- grabMSdata(sample_files[c(3, 5, 6)], grab_what="MS1")
qplotMS1data(msdata$MS1[mz%between%pmppm(118.0865)])

## End(Not run)
```

---

tmzmlMaker *Maker of tmzML documents*

Description

This function converts mzML and mzXML documents into "transposed" mzML (tmzML) documents. Traditional mass-spec data is organized by scan number, corresponding to retention time, but this isn't always the most sensible format. Often, it makes more sense to organize a mass-spec file by m/z ratio instead. This allows parsers to scan and decode a much smaller portion of the file when searching for a specific mass, as opposed to the traditional format which requires that every scan be opened, searched, and subset. The tmzML document implements this strategy and allows

the creation of MS object representations that use essentially zero memory because the data is read off the disk instead of being stored in RAM. RaMS has been designed to interface with these new file types identically to traditional files, allowing all your favorite tidyverse tricks to work just as well and much more quickly.

Usage

```
tmzmlMaker(input_filename, output_filename = NULL, verbosity = 0, binwidth = 3)
```

Arguments

input_filename	Character vector of length 1 with the name of the file to be converted. Can only handle mzML and mzXML currently - other formats should be converted to one of these first, using (for example) Proteowizard's msconvert tool.
output_filename	The name of the file that will be written out. Should end in ".tmzML" and will throw a warning otherwise. Often, it makes sense to have two folders in a working directory, one containing the original mzML files and a second, parallel folder for the tmzMLs.
verbosity	Numeric value between 0 and 2, corresponding to level of verbosity shared by the function as it proceeds. 0 means no output, 1 will produce mile markers after file opening, MS1 and MS2 conversion, and 2 will provide progress bars between each mile marker.
binwidth	Numeric value controlling the width of the bins in m/z space to create. Because MS data is created in such a way that m/z values are continuous, they must be binned together to create a discrete representation that can be searched efficiently. Lower values (0.1-1) will have faster retrieval times, while higher values (5-10) will have faster conversion times.

Value

An msdata_connection object. This object behaves exactly like a normal RaMS list with values for MS1, MS2, etc. but secretly just contains pointers to the files requested because the data is extracted on the fly. The S3 msdata_connection object is necessary to create new behaviors for '$' and '[' that allow indexing like normal.

Examples

```
## Not run:
sample_dir <- system.file("extdata", package = "RaMS")
sample_files <- list.files(sample_dir, full.names=TRUE, pattern="LB.*mzML")
tmzml_filenames <- gsub(x=sample_files, "\\.mzML.gz", ".tmzML")

# Convert a single file
tmzmlMaker(sample_files[1], tmzml_filenames[1])
file_data <- grabMSdata(tmzml_filenames[1], grab_what="everything", verbosity=2)
file_data$MS1[mz%between%pmppm(118.0865)]

# Multiple files
mapply(tmzmlMaker, sample_files, tmzml_filenames)
```

```
file_data <- grabMSdata(tmzml_filenames, grab_what="everything", verbosity=2)
betaine_data <- file_data$MS1[mz%between%pmppm(118.0865)]

# Plot output
plot(betaine_data$rt, betaine_data$int, type="l")
library(ggplot2)
ggplot(betaine_data) + geom_line(aes(x=rt, y=int, color=filename))

# Clean up afterward
file.remove(tmzml_filenames)

## End(Not run)
```

`trapz` *Trapezoidal integration of mass-spec retention time / intensity values*

Description

Performs a trapezoidal Riemann sum to calculate the area under the curve for mass-spectrometry data. Accepts a vector of retention times and the associated intensities and returns the area.

Usage

```
trapz(rts, ints, baseline = "none")
```

Arguments

`rts`	A numeric vector of retention times across an MS peak. Should be monotonically increasing and without duplicates or will throw a warning.
`ints`	A numeric vector of measured intensities across an MS peak
`baseline`	A length-1 character vector of either "none" (the default), "square", or "trapezoid".

Value

A length-1 numeric value representing the area under the curve

Examples

```
trapz(1:10, 1:10)
trapz(1:10, 10:1)

trapz(1:10, 11:20)
trapz(1:10, 11:20, baseline="square")
trapz(1:10, 11:20, baseline="trapezoid")

x_vals <- seq(-2, 2, length.out=100)
trapz(x_vals, dnorm(x_vals))
```

```
x_vals <- seq(0, pi/2, length.out=100)
trapz(x_vals, cos(x_vals))
```

[.msdata_connection *S3 indexing for msdata_connection objects*

Description

This is the step that actually performs the file opening and extraction!

Usage

```
## S3 method for class 'msdata_connection'
msdata_obj[sub_func]
```

Arguments

msdata_obj	An msdata_connection object containing files and grab_what
sub_func	The function that will be parsed and used to subset the file

Value

A data.table with columns rt, mz, int, and filename

$.msdata_connection *S3 dollar sign notation for msdata_connection objects*

Description

S3 dollar sign notation for msdata_connection objects

Usage

```
## S3 method for class 'msdata_connection'
msdata_obj$ms_level
```

Arguments

msdata_obj	An msdata_connection object containing files and grab_what
ms_level	The requested MS level of the object

Value

An msdata_connection object with only a single MS level

Index