Package 'Chromatograms'

May 1, 2026

Title Infrastructure for Chromatographic Mass Spectrometry Data

Version 1.3.0

Description The Chromatograms packages defines an efficient infrastructure for storing and handling of chromatographic mass spectrometry data. It provides different implementations of *backends* to store and represent the data. Such backends can be optimized for small memory footprint or fast data access/processing. A lazy evaluation queue and chunk-wise processing capabilities ensure efficient analysis of also very large data sets.

Depends BiocParallel, ProtGenerics (>= 1.39.2), R (>= 4.5.0)

Imports data.table, methods, S4Vectors, MsCoreUtils (>= 1.7.5), Spectra

Suggests msdata (>= 0.19.3), roxygen2, BiocStyle (>= 2.5.19), testthat, knitr (>= 1.1.0), rmarkdown, mzR (>= 2.41.4), MsBackendMetaboLights (>= 1.3.1), pheatmap, vdiffr, IRanges, RColorBrewer

License Artistic-2.0

Encoding UTF-8

VignetteBuilder knitr

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

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

biocViews Infrastructure, Metabolomics, MassSpectrometry, Proteomics

Roxygen list(markdown=TRUE)

RoxygenNote 7.3.3

Collate 'AllGenerics.R' 'ChromBackend-functions.R' 'ChromBackend.R' 'hidden_aliases.R' 'helpers.R' 'ChromBackendMemory.R' 'ChromBackendMzR.R' 'ChromBackendSpectra.R' 'Chromatograms.R' 'Chromatograms-chromData.R' 'Chromatograms-peaksData.R' 'Chromatograms-plot.R' 'Chromatograms-processingQueue.R'

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

git_branch devel

git_last_commit 31a1ba2

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-05-01

Author Johannes Rainer [aut] (ORCID: <https://orcid.org/0000-0002-6977-7147>),
Laurent Gatto [aut] (ORCID: <https://orcid.org/0000-0002-1520-2268>),
Philippine Louail [aut, cre] (ORCID:
<https://orcid.org/0009-0007-5429-6846>, fnd: European Union
HORIZON-MSCA-2021 project Grant No. 101073062: HUMAN – Harmonising
and Unifying Blood Metabolic Analysis Networks)

Maintainer Philippine Louail <philippine.louail@outlook.com>

Contents

`Chromatograms` *The Chromatograms class to manage and access chromatographic data*

Description

The `Chromatograms` class encapsules chromatographic data and related metadata. The chromatographic data is represented by a *backend* extending the virtual ChromBackend class which provides the raw data to the `Chromatograms` object. Different backends and their properties are described in the ChromBackend class documentation.

Available Backends: The package provides several backends:

- `ChromBackendMemory`: Stores data in memory (default, ideal for small datasets).
- `ChromBackendMzR`: Reads peaks data from raw MS files on demand.
- `ChromBackendSpectra`: Generates chromatographic data from a `Spectra` object. This backend supports both in-memory and file-backed `Spectra` objects, using an internal `spectraSortIndex` to avoid physically reordering the spectra.

Usage

```
## S4 method for signature 'ChromBackendOrMissing'
Chromatograms(object = ChromBackendMemory(), processingQueue = list(), ...)

## S4 method for signature 'Spectra'
Chromatograms(
```

```
  object,
  summarize.method = c("sum", "max"),
  chromData = data.frame(),
  factorize.by = c("msLevel", "dataOrigin"),
  spectraVariables = character(),
  ...
)

## S4 method for signature 'Chromatograms,ChromBackend'
setBackend(
  object,
  backend,
  f = processingChunkFactor(object),
  BPPARAM = SerialParam(),
  ...
)

## S4 method for signature 'Chromatograms'
x$name

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

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

## S4 method for signature 'Chromatograms'
x[[i, j, ...]]

## S4 replacement method for signature 'Chromatograms'
x[[i, j, ...]] <- value

## S4 method for signature 'Chromatograms'
factorize(object, factorize.by = c("msLevel", "dataOrigin"), ...)

## S4 method for signature 'Chromatograms'
chromExtract(object, peak.table, by, ...)

## S4 method for signature 'Chromatograms'
filterEmptyChromatograms(object, ...)
```

Arguments

`object`	A Chromatograms object.
`processingQueue`	list a list of processing steps (i.e. functions) to be applied to the chromatographic data. The processing steps are applied in the order they are listed in the `processingQueue`.
`...`	Additional arguments.
`summarize.method`	For `Chromatograms` created with a `Spectra` object: A `character` vector with the name of the function to be used to summaries the spectra data intensity. The

available methods are "sum" and "max". The default is "sum".

chromData — For `Chromatograms()` build from a `Spectra` object backend, a `data.frame` with the chromatographic data. If not provided (or if empty), a default `data.frame` with the core chromatographic variables will be created.

factorize.by — A `character` vector with the names of the variables in the `Spectra` object and the `chromData` slot that should be used to factorize the `Spectra` object data to generate the chromatographic data.

spectraVariables — A `character` vector specifying which variables from the `Spectra` object should be added to the chromData. These will be mapped using the `chromSpectraIndex` variable.

backend — ChromBackend object providing the raw data for the `Chromatograms` object.

f — `factor` defining the grouping to split the `Chromatograms` object.

BPPARAM — Parallel setup configuration. See `BiocParallel::bpparam()` for more information.

x — A Chromatograms object.

name — A `character` string specifying the name of the variable to access.

value — The value to replace the variable with.

i — For `[`: `integer`, `logical` or `character` to subset the object.

j — For `[` and `[[`: ignored.

drop — For `[`: `logical(1)` default to `FALSE`.

peak.table — For `chromExtract()` A `data frame` containing the following minimum columns: - rtMin: Minimum retention time for each peak. Cannot be NA. - rtMax: Maximum retention time for each peak. Cannot be NA. - mzMin: Minimum m/z value for each peak. - mzMax: Maximum m/z value for each peak. Additionally, the `peak.table` must include columns that uniquely identify chromatograms in the `object`. Common choices are "msLevel" and/or "dataOrigin". These columns must also be present in the `chromData` of the `object`. Any extra columns in `peak.table` will be added to the `chromData` of the newly created object.

by — A `character` vector naming one or more columns that uniquely identify chromatograms in both `peak.table` and `chromData(object)`. The combination of these columns must be unique within `chromData(object)`. Typically includes `"dataOrigin"`, `"msLevel"`, or both.

Value

Refer to the individual function description for information on the return value.

Creation of objects

`Chromatograms` objects can be created using the `Chromatograms()` construction function. Either by providing a `ChromBackend` object or by providing a `Spectra` object. The `Spectra` object will be used to generate a `Chromatograms` object with a backend of class `ChromBackendSpectra`.

Data stored in a `Chromatograms` object

The `Chromatograms` object is a container for chromatographic data, which includes peaks data (*retention time* and related intensity values, also referred to as *peaks data variables* in the context of `Chromatograms`) and metadata of individual chromatogram (so called *chromatograms variables*). While a core set of chromatograms variables (the `coreChromatogramsVariables()`) and peaks data variables (the `corePeaksVariables()`) are guaranteed to be provided by a `Chromatograms`, it is possible to add arbitrary variables to a `Chromatograms` object.

The `Chromatograms` object is designed to contain chromatographic data of a (large) set of chromatograms. The data is organized *linearly* and can be thought of a list of chromatograms, i.e. each element in the `Chromatograms` is one chromatogram.

The *chromatograms variables* information in the `Chromatograms` object can be accessed using the `chromData()` function. Specific chromatograms variables can be accessed by either precising the `"columns"` parameter in `chromData()` or using `$`. `@chromData` can be accessed, replaced but also filtered/subsetted. Refer to the chromData documentation for more details.

The *peaks data variables* information in the `Chromatograms` object can be accessed using the `peaksData()` function. Specific peaks variables can be accessed by either precising the `"columns"` parameter in `peaksData()` or using `$`. `@peaksData` can be accessed, replaced but also filtered/subsetted. Refer to the peaksData documentation for more details.

Processing of `Chromatograms` objects

Functions that process the chromatograms data in some ways can be applied to the object either directly or by using the `processingQueue` mechanism. The `@processingQueue` is a list of processing steps that are stored within the object and only applied when needed. This was created so that the data can be processed in a single step and is very useful for larger datasets. This is even more true as this processing queue will call function that can be applied on the data in a chunk-wise manner. This allows for parallel processing of the data and reduces the memory demand. To read more about the `processingQueue`, and how to parallelize your processes, see the processingQueue documentation.

Subsetting and accessing data

The `Chromatograms` class supports subsetting by chromatogram (i.e. rows) using the `[` operator. The `[` operator does not support subsetting by columns. Specific chromatograms or peaks variables can be accessed using the `[[` operator or the `$` operator. The `[[` operator can also be used to replace specific chromatograms or peaks variables.

Changing the backend

The `setBackend()` function can be used to change the backend of a `Chromatograms` object. This can be useful to switch to a backend that better suits the needs of the user, for example switching to a memory-based backend for smaller datasets or to a file-based backend for larger datasets. The `setBackend()` function supports parallelization of the backend conversion using the `BPPARAM` parameter. Note that any queued processing steps are applied during the backend switch (since `peaksData()` is called to transfer the data) and the processing queue is emptied afterwards.

Filtering chromatograms

- `filterEmptyChromatograms()`: removes empty chromatograms (i.e. chromatograms without peaks) from the object. Returns the filtered `Chromatograms` object (with chromatograms in their original order).

Extracting chromatograms based on a peak table

The `chromExtract()` function allows users to extract specific regions of interest from a `Chromatograms` object based on a user-provided peak table. Each row in the `peak.table` defines a region to extract, using minimum and maximum retention time (and m/z in the case of `chromBackendSpectra`) boundaries, and identifiers that uniquely match chromatograms in the object.

The resulting **new** `Chromatograms` object contains only chromatograms overlapping the specified regions, with updated metadata reflecting the extracted boundaries.

This function is most commonly used to subset chromatographic data around detected peaks or predefined time/mass ranges, for example to reprocess, visualize, or quantify extracted chromatograms corresponding to known features. It's important to notes that filtering by m/z is only supported when using a `ChromBackendSpectra` backend. if the `mzMin` and `mzMax` columns are provided when using other backends, they will be ignored.

Note

This needs to be discussed, if we want for example to be able to set a a backend to `ChromBackendMzR` we need to implement backendInitialize() better. = Support peaksData and chromData as arguments AND have a way to write .mzml files (which we do not have for chromatographic data).

See Also

chromData for a general description of the chromatographic metadata available in the object, as well as how to access, replace and subset them. peaksData for a general description of the chromatographic peaks data available in the object, as well as how to access, replace and subset them. processingQueue for more information on the queuing of processings and parallelization for larger dataset.

Examples

```
## Create a Chromatograms object with ChromBackendMemory
cdata <- data.frame(
    msLevel = c(1L, 1L, 1L),
    mz = c(112.2, 123.3, 134.4),
    dataOrigin = c("mem1", "mem2", "mem3")
)
pdata <- list(
    data.frame(rtime = c(2.1, 2.5, 3.0, 3.4, 3.9),
               intensity = c(100, 250, 400, 300, 150)),
    data.frame(rtime = c(3.5, 4.0, 4.5),
               intensity = c(80, 120, 90)),
    data.frame(rtime = c(5.1, 5.8, 6.3, 6.9, 7.5),
               intensity = c(80, 500, 1200, 600, 120))
)
chr <- Chromatograms(ChromBackendMemory(), chromData = cdata, peaksData = pdata)
chr

## Create a Chromatograms object from a Spectra object
library(MsBackendMetaboLights)
library(Spectra)

be <- backendInitialize(MsBackendMetaboLights(),
    mtblsId = "MTBLS39",
    filePattern = c("63B.cdf")
)
```

```
s <- Spectra(be)
s <- setBackend(s, MsBackendMemory())
chr <- Chromatograms(s)

## Subset
chr[1:2]

## Access a specific variable
chr[["msLevel"]]
chr$msLevel

## Replace data of a specific variable
chr$msLevel <- c(2L, 2L, 2L)

## Re-factorize the data
chr <- factorize(chr)

## Change the backend to memory
chr <- setBackend(chr, ChromBackendMemory())
```

ChromBackendMemory *Improved in-memory Chromatographic data backend*

Description

ChromBackendMemory: This backend stores chromatographic data directly in memory, making it ideal for small datasets or testing. It can be initialized with a data.frame of chromatographic data via the chromData parameter and a list of data.frame entries for peaks data using the peaksData parameter. These data can be accessed with the chromData() and peaksData() functions.

Usage

```
ChromBackendMemory()

## S4 method for signature 'ChromBackendMemory'
backendInitialize(
  object,
  chromData = fillCoreChromVariables(data.frame()),
  peaksData = list(.EMPTY_PEAKS_DATA),
  ...
)
```

Arguments

object	A ChromBackendMemory object.
chromData	For backendInitialize() of a ChromBackendMemory backend, a data.frame with the chromatographic data. If not provided (or if empty), a default data.frame with the core chromatographicvariables will be created.
peaksData	For backendInitialize() of a ChromBackendMemory backend, a list of data.frame with the peaks data. If not provided (or if empty), a default list of empty data.frame with the core peaks variables will be created. The length of the list should match the number of chromatograms in the chromData parameter.
...	Additional parameters to be passed.

Value

Refer to the individual function description for information on the return value.

Author(s)

Philippine Louail

Examples

```
## Method 1: Initialize backend directly
cdata <- data.frame(
    msLevel = c(1L, 1L, 1L),
    mz = c(112.2, 123.3, 134.4),
    dataOrigin = c("mem1", "mem2", "mem3")
)

pdata <- list(
    data.frame(
        rtime = c(12.4, 12.8, 13.2, 14.6),
        intensity = c(123.3, 153.6, 2354.3, 243.4)
    ),
    data.frame(
        rtime = c(45.1, 46.2),
        intensity = c(100, 80.1)
    ),
    data.frame(
        rtime = c(12.4, 12.8, 13.2, 14.6),
        intensity = c(123.3, 153.6, 2354.3, 243.4)
    )
)

cbm <- ChromBackendMemory()
cbm <- backendInitialize(cbm, chromData = cdata, peaksData = pdata)
cbm

## Method 2: Use Chromatograms constructor (recommended)
chr <- Chromatograms(ChromBackendMemory(), chromData = cdata, peaksData = pdata)
chr
```

ChromBackendMzR *Chromatographic Data Backend for Reading mzML Files*

Description

The ChromBackendMzR inherits all slots and methods from the base ChromBackendMemory backend, providing additional functionality for reading chromatographic data from mzML files.

Unlike the ChromBackendMemory backend, the ChromBackendMzR backend should have the *dataOrigin* chromatographic variables populated with the file path of the mzML file from which the chromatographic data was read.

Note that the ChromBackendMzR backend is read-only and does not support direct modification of chromatographic data. However, it does support peaksData slot replacement, which will modify

the @peaksData slot but not the local mzML files. This is indicated by the "inMemory" slot being set to TRUE.

Implementing functionalities with the ChromBackendMzR backend should be simplified as much as possible and reuse the methods already implemented for ChromBackendMemory when possible.

Usage

```
ChromBackendMzR()

## S4 method for signature 'ChromBackendMzR'
backendInitialize(object, files = character(), BPPARAM = bpparam(), ...)
```

Arguments

object	A ChromBackendMzR object.
files	A character vector of file paths to mzML files.
BPPARAM	Parallel setup configuration. See BiocParallel::bpparam() for more information.
...	Additional parameters to be passed.

Value

Refer to the individual function description for information on the return value.

Author(s)

Philippine Louail

Examples

```
library(mzR)
library(msdata)

## Load an mzML file
MRM_file <- system.file("proteomics", "MRM-standmix-5.mzML.gz",
    package = "msdata"
)

## Initialize the ChromBackendMzR object
be_empty <- ChromBackendMzR()
be <- backendInitialize(be_empty, files = MRM_file, BPPARAM = SerialParam())
```

ChromBackendSpectra *Chromatographic Data Backend for Spectra Objects*

Description

The ChromBackendSpectra class extends ChromBackendMemory, inheriting all its slots and methods while providing additional functionality for summarizing chromatographic data from Spectra::Spectra() objects.

It can be initialized with a Spectra object, which is stored in the spectra slot of the backend. Users can also provide a data.frame containing chromatographic metadata, stored in @chromData. This metadata filters the Spectra object and generates peaksData. If chromData is not provided, a default data.frame is created from the Spectra data. An "rtMin", "rtMax", "mzMin", and "mzMax" column will be created by condensing the Spectra data corresponding to each unique combination of the factorize.by variables.

By "factorization" we mean the process of grouping spectra into chromatograms based on specified variables. For example, using factorize.by = c("msLevel", "dataOrigin") means that all MS1 spectra from file "A" form one chromatogram, all MS2 spectra from file "A" form another, and so on. Each unique combination of the factorization variables creates a separate chromatogram. This is essential for organizing spectral data into meaningful chromatographic traces that can be visualized and analyzed.

The *dataOrigin* core chromatogram variable should reflect the *dataOrigin* of the Spectra object. The factorize.by parameter defines the variables for grouping Spectra data into chromatographic data. The default is c("msLevel", "dataOrigin"), which will define separate chromatograms for each combination of msLevel and dataOrigin. These variables must be in both the spectraData() of the Spectra and chromData (if provided).

The summarize.method parameter defines how spectral data intensity is summarized:

- **"sum"**: Sums intensity to create a Total Ion Chromatogram (TIC).
- **"max"**: Takes max intensity for a Base Peak Chromatogram (BPC).

If chromData or its factorization columns are modified, the factorize() method must be called to update chromSpectraIndex.

Usage

```
ChromBackendSpectra()

## S4 method for signature 'ChromBackendSpectra'
backendInitialize(
  object,
  spectra = Spectra(),
  factorize.by = c("msLevel", "dataOrigin"),
  summarize.method = c("sum", "max"),
  chromData = fillCoreChromVariables(),
  spectraVariables = character(),
  ...
)

chromSpectraIndex(object)
```

Arguments

object	A ChromBackendSpectra object.
spectra	A Spectra object.

factorize.by A character vector of spectraVariables for grouping Spectra data into chromatographic data (i.e., creating separate chromatograms for each unique combination of these variables). Default: c("msLevel", "dataOrigin"), which creates one chromatogram per MS level per data file. If chromData is provided, it **must** also contain these columns.

summarize.method
A character string specifying intensity summary: "sum" (default) or "max".

chromData A data.frame with chromatographic data for use in backendInitialize(). If missing, a default is generated. Columns like rtMin, rtMax, mzMin, and mzMax must be provided and not contain NA values. Use -Inf/Inf for unspecified values. The "dataOrigin" column must match the Spectra object's "dataOrigin".

spectraVariables
A character vector specifying which variables from the Spectra object should be added to the chromData. These will be mapped using the chromSpectraIndex variable.

... Additional parameters.

Details

No peaksData is stored until the user calls a function that generates it (e.g., rtime(), peaksData(), intensity()). The @peaksData slot replacement is unsupported — modifications are temporary to optimize memory. The @inMemory slot indicates this with TRUE.

Spectra Sort Index: The ChromBackendSpectra backend maintains a spectraSortIndex slot that stores a sort order for the internal Spectra object based on dataOrigin and rtime. To optimize performance, the sort index is only computed and stored when the spectra are unsorted; if already sorted (which is typical for most real-world data), spectraSortIndex remains empty (integer()). This avoids unnecessary subsetting operations. The sort index is automatically recalculated whenever the factorize() method is called, ensuring it remains valid and consistent. This approach avoids the need to physically reorder disk-backed Spectra objects, which would require loading all data into memory.

Factorize and Subsetting: The factorize() method updates the chromSpectraIndex in both chromData and the @spectra to reflect the current grouping, and recalculates spectraSortIndex to maintain the correct sort order. The [subsetting operator properly handles subsetting of both @chromData, @peaksData, and @spectra, while updating the spectraSortIndex to reference valid positions in the subsetted data.

ChromBackendSpectra should reuse ChromBackendMemory methods whenever possible to keep implementations simple.

Value

Refer to the individual function description for information on the return value.

Author(s)

Philippine Louail, Johannes Rainer.

Examples

```
library(Spectra)
library(MsBackendMetaboLights)
```

```
## Get Spectra data from MetaboLights
be <- backendInitialize(MsBackendMetaboLights(),
    mtblsId = "MTBLS39",
    filePattern = c("63B.cdf")
)
s <- Spectra(be)

s <- setBackend(s, MsBackendMemory())

## Initialize ChromBackendSpectra
be_empty <- new("ChromBackendSpectra")
be <- backendInitialize(be_empty, s)

## replace the msLevel data
msLevel(be) <- c(1L, 2L, 3L)

## re-factorize the data
be <- factorize(be)

## Create BPC : we summarize the intensity present in the Spectra object
## by the maximum value, thus creating a Base Peak Chromatogram.
be <- backendInitialize(be_empty, s, summarize.method = "max")

## Can now see the details of this bpc by looking at the chromData of our
## object
chromData(be)

## Another possibilities is to create eics from the Spectra object.
## Here we create an EIC with a specific m/z and retention time window.
df <- data.frame(mzMin = 100.01, mzMax = 100.02 , rtMin = 50, rtMax = 100)
be <- backendInitialize(be_empty, s, summarize.method = "sum")
chromData(be) <- cbind(chromData(be), df)

## now when we call the peaksData function, we will get the intensity
## of the spectra object that are in the m/z and retention time window
## defined in the chromData.
peaksData(be)
```

`chromData` *Chromatographic Peaks Metadata.*

Description

As explained in the `Chromatograms` class documentation, the `Chromatograms` object is a container for chromatogram data that includes chromatographic peaks data (*retention time* and related intensity values, also referred to as *peaks data variables* in the context of `Chromatograms`) and metadata of individual chromatograms (so called *chromatograms variables*).

The *chromatograms variables* information can be accessed using the `chromData()` function. it is also possible to access specific chromatograms variables using `$`.
`@chromData` can be accessed, replaced but also filtered/subsetted. Refer to the sections below for more details.

Usage

```
## S4 method for signature 'Chromatograms'
chromData(object, columns = chromVariables(object), drop = FALSE)

## S4 replacement method for signature 'Chromatograms'
chromData(object) <- value

## S4 method for signature 'Chromatograms'
chromVariables(object)

## S4 method for signature 'Chromatograms'
chromIndex(object)

## S4 replacement method for signature 'Chromatograms'
chromIndex(object) <- value

## S4 method for signature 'Chromatograms'
collisionEnergy(object)

## S4 replacement method for signature 'Chromatograms'
collisionEnergy(object) <- value

## S4 method for signature 'Chromatograms'
dataOrigin(object)

## S4 replacement method for signature 'Chromatograms'
dataOrigin(object) <- value

## S4 method for signature 'Chromatograms'
msLevel(object)

## S4 replacement method for signature 'Chromatograms'
msLevel(object) <- value

## S4 method for signature 'Chromatograms'
mz(object)

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

## S4 method for signature 'Chromatograms'
mzMax(object)

## S4 replacement method for signature 'Chromatograms'
mzMax(object) <- value

## S4 method for signature 'Chromatograms'
mzMin(object)

## S4 replacement method for signature 'Chromatograms'
mzMin(object) <- value
```

```
## S4 method for signature 'Chromatograms'
length(x)

## S4 method for signature 'Chromatograms'
precursorMz(object)

## S4 replacement method for signature 'Chromatograms'
precursorMz(object) <- value

## S4 method for signature 'Chromatograms'
precursorMzMin(object)

## S4 replacement method for signature 'Chromatograms'
precursorMzMin(object) <- value

## S4 method for signature 'Chromatograms'
precursorMzMax(object)

## S4 replacement method for signature 'Chromatograms'
precursorMzMax(object) <- value

## S4 method for signature 'Chromatograms'
productMz(object)

## S4 replacement method for signature 'Chromatograms'
productMz(object) <- value

## S4 method for signature 'Chromatograms'
productMzMin(object)

## S4 replacement method for signature 'Chromatograms'
productMzMin(object) <- value

## S4 method for signature 'Chromatograms'
productMzMax(object)

## S4 replacement method for signature 'Chromatograms'
productMzMax(object) <- value

## S4 method for signature 'Chromatograms'
filterChromData(
  object,
  variables = character(),
  ranges = numeric(),
  match = c("any", "all"),
  keep = TRUE
)
```

Arguments

`object`	A Chromatograms object.
`columns`	A `character` vector of chromatograms variables to extract.

drop	A logical indicating whether to drop dimensions when extracting a single variable.
value	replacement value for <- methods. See individual method description or expected data type.
x	A Chromatograms object.
variables	For filterChromData(): character vector with the names of the chromatogram variables to filter for. The list of available chromatogram variables can be obtained with chromVariables().
ranges	For filterChromData() : a numeric vector of paired values (upper and lower boundary) that define the ranges to filter the object. These paired values need to be in the same order as the variables parameter (see below).
match	For filterChromData() : character(1) defining whether the condition has to match for all provided ranges (match = "all"; the default), or for any of them (match = "any") for chromatogram data to be retained.
keep	For filterChromData(): logical(1) defining whether to keep (keep = TRUE) or remove (keep = FALSE) the chromatogram data that match the condition.

Value

Refer to the individual function description for information on the return value.

Chromatograms variables and accessor functions

The following chromatograms variables are guaranteed to be provided by a Chromatograms object and to be accessible with either the chromData() or a specific function named after the variables names:

- chromIndex: an integer with the index of the chromatogram in the original source file (e.g. *mzML* file).
- collisionEnergy: for SRM data, numeric with the collision energy of the precursor.
- dataOrigin: optional character with the origin of the data.
- msLevel: integer defining the MS level of the data.
- mz: optional numeric with the (target) m/z value for the chromatographic data.
- mzMin: optional numeric with the lower m/z value of the m/z range in case the data (e.g. an extracted ion chromatogram EIC) was extracted from a Spectra object.
- mzMax: optional numeric with the upper m/z value of the m/z range.
- precursorMz: for SRM data, numeric with the target m/z of the precursor (parent).
- precursorMzMin: for SRM data, optional numeric with the lower m/z of the precursor's isolation window.
- precursorMzMax: for SRM data, optional numeric with the upper m/z of the precursor's isolation window.
- productMz for SRM data, numeric with the target m/z of the product ion.
- productMzMin: for SRM data, optional numeric with the lower m/z of the product's isolation window.
- productMzMax: for SRM data, optional numeric with the upper m/z of the product's isolation window.

Filter Chromatograms variables

Functions that filter `Chromatograms` based on chromatograms variables (i.e, `@chromData`) will remove chromatographic data that do not meet the specified conditions. This means that if a chromatogram is filtered out, its corresponding `@chromData` and `@peaksData` will be removed from the object immediately.

The available functions to filter chromatogram data are:

- `filterChromData()`: Filters numerical chromatographic data variables based on the provided numerical `ranges`. The method returns a `Chromatograms` object containing only the chromatograms that match the specified conditions. This function results in an object with fewer chromatograms than the original.

Author(s)

Philippine Louail

See Also

Chromatograms for a general description of the `Chromatograms` object. peaksData for a general description of the chromatographic peaks data available in the object, as well as how to access, replace and subset them. processingQueue for more information on the queuing of processings and parallelization for larger dataset processing.

Examples

```
# Create a Chromatograms object
cdata <- data.frame(
    msLevel = c(1L, 1L, 1L),
    mz = c(112.2, 123.3, 134.4),
    chromIndex = c(1L, 2L, 3L)
)

be <- backendInitialize(new("ChromBackendMemory"), chromData = cdata)

chr <- Chromatograms(be)

# Access chromatograms variables
chromData(chr)

# Access specific chromatograms variables
chromData(chr, columns = "msLevel")

msLevel(chr)

# Replace chromatograms variables
msLevel(chr) <- c(1L, 2L, 2L)

# Filter chromatograms variables
filterChromData(chr,
    variables = "msLevel", ranges = c(1L, 1L),
    keep = FALSE
)
```

concatenateChromatograms *Merging, combining and splitting Chromatograms*

Description

Various functions are available to combine or split data from one or more Chromatograms objects. These are:

- c() and concatenateChromatograms(): combines several Chromatograms objects into a single object. The resulting Chromatograms contains all data from all individual Chromatograms, i.e. the union of all their chromatograms variables. Concatenation will fail if the processing queue of any of the Chromatograms objects is not empty or if different backends are used for the Chromatograms objects. In such cases it is suggested to first change the backends of all Chromatograms to the same type of backend (using the ProtGenerics::setBackend() function) and to eventually (if needed) apply the processing queue using the ProtGenerics::applyProcessing() function.
- split(): splits the Chromatograms object based on a provided grouping factor returning a list of Chromatograms objects.

Usage

```
concatenateChromatograms(x, ...)

## S4 method for signature 'Chromatograms'
c(x, ...)

## S4 method for signature 'Chromatograms,ANY'
split(x, f, drop = FALSE, ...)
```

Arguments

x	A Chromatograms object.
...	For c() and concatenateChromatograms(): Chromatograms objects or a list of Chromatograms objects.
f	factor defining the grouping to split the Chromatograms object.
drop	For split(): not considered.

Value

- c() and concatenateChromatograms(): a single Chromatograms object containing the data from all input objects.
- split(): a list of Chromatograms objects.

Examples

```
## Create two Chromatograms objects
cdata1 <- data.frame(
    msLevel = c(1L, 1L),
    mz = c(112.2, 123.3),
```

```
    dataOrigin = c("file1", "file1")
)
pdata1 <- list(
    data.frame(rtime = c(1.0, 2.0, 3.0), intensity = c(100, 200, 150)),
    data.frame(rtime = c(1.0, 2.0, 3.0), intensity = c(80, 120, 90))
)
chr1 <- Chromatograms(
    ChromBackendMemory(),
    chromData = cdata1,
    peaksData = pdata1
)

cdata2 <- data.frame(
    msLevel = c(2L, 2L),
    mz = c(134.4, 145.5),
    dataOrigin = c("file2", "file2")
)
pdata2 <- list(
    data.frame(rtime = c(4.0, 5.0, 6.0), intensity = c(300, 400, 350)),
    data.frame(rtime = c(4.0, 5.0, 6.0), intensity = c(200, 250, 180))
)
chr2 <- Chromatograms(
    ChromBackendMemory(),
    chromData = cdata2,
    peaksData = pdata2
)

## Combine using c()
chr_combined <- c(chr1, chr2)
chr_combined

## Combine using concatenateChromatograms
chr_combined2 <- concatenateChromatograms(chr1, chr2)

## Combine a list of Chromatograms
chr_list <- list(chr1, chr2)
chr_combined3 <- concatenateChromatograms(chr_list)

## Split by msLevel
chr_split <- split(chr_combined, f = chr_combined$msLevel)
chr_split
```

`coreChromVariables` *Chromatographic MS Data Backends*

Description

`ChromBackend` is a virtual class that defines what different backends need to provide to be used by the `Chromatograms` package and classes.

The backend should provide access to the chromatographic data which mainly consists of (paired) intensity and retention time values. Additional chromatographic metadata such as MS level and precursor and product m/z should also be provided.

Through their implementation different backends can be either optimized for minimal memory requirements or performance. Each backend needs to implement data access methods listed in section *Backend functions:* below.

And example implementation and more details and descriptions are provided in the *Creating new* `ChromBackend` *classes for Chromatograms* vignette.

Currently available backends are:

- `ChromBackendMemory`: This backend stores chromatographic data directly in memory, making it ideal for small datasets or testing. It can be initialized with a `data.frame` of chromatographic data via the `chromData` parameter and a `list` of `data.frame` entries for peaks data using the `peaksData` parameter. These data can be accessed with the `chromData()` and `peaksData()` functions.
- `ChromBackendMzR`: The `ChromBackendMzR` inherits all slots and methods from the base `ChromBackendMemory` backend, providing additional functionality for reading chromatographic data from mzML files.
- `ChromBackendSpectra`: The `ChromBackendSpectra` inherits all slots and methods from the base `ChromBackendMemory` backend, providing additional functionality for reading chromatographic data from `Spectra` objects.

Filter the peak data based on the provided ranges for the given variables.

Usage

```
coreChromVariables()

corePeaksVariables()

## S4 method for signature 'ChromBackend'
x$name

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

## S4 method for signature 'ChromBackend'
backendMerge(object, ...)

## S4 method for signature 'ChromBackend'
chromData(object, columns = chromVariables(object), drop = FALSE)

## S4 replacement method for signature 'ChromBackend'
chromData(object) <- value

## S4 method for signature 'ChromBackend'
chromExtract(object, peak.table, by)

## S4 method for signature 'ChromBackend'
factorize(object, ...)

## S4 method for signature 'ChromBackend'
peaksData(object, columns = c("rtime", "intensity"), drop = FALSE, ...)

## S4 replacement method for signature 'ChromBackend'
```

```
peaksData(object) <- value

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

## S4 method for signature 'ChromBackend'
x[[i, j, ...]]

## S4 replacement method for signature 'ChromBackend'
x[[i, j, ...]] <- value

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

## S4 method for signature 'ChromBackend'
backendInitialize(object, ...)

## S4 method for signature 'ChromBackend'
backendParallelFactor(object, ...)

## S4 method for signature 'list'
backendMerge(object, ...)

## S4 method for signature 'ChromBackend'
chromIndex(object)

## S4 replacement method for signature 'ChromBackend'
chromIndex(object) <- value

## S4 method for signature 'ChromBackend'
chromVariables(object)

## S4 method for signature 'ChromBackend'
collisionEnergy(object)

## S4 replacement method for signature 'ChromBackend'
collisionEnergy(object) <- value

## S4 method for signature 'ChromBackend'
dataOrigin(object)

## S4 replacement method for signature 'ChromBackend'
dataOrigin(object) <- value

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

## S4 method for signature 'ChromBackend,missing'
extractByIndex(object, i)

## S4 method for signature 'ChromBackend'
intensity(object)
```

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

## S4 method for signature 'ChromBackend'
isEmpty(x)

## S4 method for signature 'ChromBackend'
isReadOnly(object)

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

## S4 method for signature 'ChromBackend'
lengths(x)

## S4 method for signature 'ChromBackend'
msLevel(object)

## S4 replacement method for signature 'ChromBackend'
msLevel(object) <- value

## S4 method for signature 'ChromBackend'
mz(object)

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

## S4 method for signature 'ChromBackend'
mzMax(object)

## S4 replacement method for signature 'ChromBackend'
mzMax(object) <- value

## S4 method for signature 'ChromBackend'
mzMin(object)

## S4 replacement method for signature 'ChromBackend'
mzMin(object) <- value

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

## S4 method for signature 'ChromBackend'
precursorMz(object)

## S4 replacement method for signature 'ChromBackend'
precursorMz(object) <- value

## S4 method for signature 'ChromBackend'
precursorMzMax(object)
```

```
## S4 replacement method for signature 'ChromBackend'
precursorMzMax(object) <- value

## S4 method for signature 'ChromBackend'
precursorMzMin(object)

## S4 replacement method for signature 'ChromBackend'
precursorMzMin(object) <- value

## S4 method for signature 'ChromBackend'
productMz(object)

## S4 replacement method for signature 'ChromBackend'
productMz(object) <- value

## S4 method for signature 'ChromBackend'
productMzMax(object)

## S4 replacement method for signature 'ChromBackend'
productMzMax(object) <- value

## S4 method for signature 'ChromBackend'
productMzMin(object)

## S4 replacement method for signature 'ChromBackend'
productMzMin(object) <- value

## S4 method for signature 'ChromBackend'
reset(object)

## S4 method for signature 'ChromBackend'
rtime(object)

## S4 replacement method for signature 'ChromBackend'
rtime(object) <- value

## S4 method for signature 'ChromBackend,ANY'
split(x, f, drop = FALSE, ...)

## S4 method for signature 'ChromBackend'
filterChromData(
  object,
  variables = character(),
  ranges = numeric(),
  match = c("any", "all"),
  keep = TRUE
)

## S4 method for signature 'ChromBackend'
filterEmptyChromatograms(object, ...)

## S4 method for signature 'ChromBackend'
```

```
filterPeaksData(
  object,
  variables = character(),
  ranges = numeric(),
  match = c("any", "all"),
  keep = TRUE
)

## S4 method for signature 'ChromBackend'
supportsSetBackend(object, ...)

## S4 method for signature 'ChromBackend'
imputePeaksData(
  object,
  method = c("linear", "spline", "gaussian", "loess"),
  span = 0.3,
  sd = 1,
  window = 2,
  extrapolate = FALSE,
  ...
)
```

Arguments

x	Object extending ChromBackend.
name	For $ and $<-: the name of the chromatogram variable to return or set.
value	Replacement value for <- methods. See individual method description or expected data type.
object	Object extending ChromBackend.
...	Additional arguments.
columns	For chromData() accessor: optional character with column names (chromatogram variables) that should be included in the returned data.frame. By default, all columns are returned.
drop	For chromData() and peaksData(): logical(1) default to FALSE. If TRUE, and one column is requested by the user, the method should return a vector (or list of vector for peaksData()) of the single column requested.
peak.table	For chromExtract() A data frame containing the following minimum columns: - rtMin: Minimum retention time for each peak. Cannot be NA. - rtMax: Maximum retention time for each peak. Cannot be NA. - mzMin: Minimum m/z value for each peak. - mzMax: Maximum m/z value for each peak. Additionally, the peak.table must include columns that uniquely identify chromatograms in the object. Common choices are "msLevel" and/or "dataOrigin". These columns must also be present in the chromData of the object. Any extra columns in peak.table will be added to the chromData of the newly created object.
by	for chromExtract() A character vector specifying one or more column names that are present in both peak.table and chromData(object). These columns uniquely identify chromatograms. The combination of these columns must be unique in chromData(object). Can be of length 1 or greater.
i	For [: integer, logical or character to subset the object.

j	For [and [[: ignored.
BPPARAM	Parallel setup configuration. See BiocParallel::bpparam() for more information.
f	factor defining the grouping to split x. See split().
variables	For filterChromData(): character vector with the names of the chromatogram variables to filter for. The list of available chromatogram variables can be obtained with chromVariables().
ranges	For filterChromData() : a numeric vector of paired values (upper and lower boundary) that define the ranges to filter the object. These paired values need to be in the same order as the variables parameter (see below).
match	For filterChromData() : character(1) defining whether the condition has to match for all provided ranges (match = "all"; the default), or for any of them (match = "any") for chromatogram data to be retained.
keep	For filterChromData(): logical(1) defining whether to keep (keep = TRUE) or remove (keep = FALSE) the chromatogram data that match the condition.
method	For imputePeaksData(): character(1): Imputation method ("linear", "spline", "gaussian", "loess")
span	For imputePeaksData: numeric(1), for the loess method: Smoothing parameter (only used if method == "loess")
sd	For imputePeaksData: numeric(1), for the gaussian method: Standard deviation for Gaussian kernel (only used if method == "gaussian")
window	For imputePeaksData: integer, for the gaussian method: Half-width of Gaussian kernel window (e.g., 2 gives window size 5)
extrapolate	For imputePeaksData: logical(1) (default FALSE). If TRUE, missing values at the beginning and end of a chromatogram (outside the range of observed values) will be extrapolated. If FALSE, only interpolation is performed and leading/trailing NA values remain NA.

Value

Refer to the individual function description for information on the return value.

Core chromatogram variables

The *core* chromatogram variables are variables (metadata) that can/should be provided by a backend. For each of these variables a value needs to be returned, if none is defined, a missing value (of the correct data type) should be returned. The names of the chromatogram variables in your current chromatogram object are returned with the chromVariables() function.

For each core chromatogram variable a dedicated access method exists. In contrast to the peaks data described below, a single value should be returned for each chromatogram.

The coreChromVariables() function returns the core chromatogram variables along with their expected (defined) data type.

The core chromatogram variables (in alphabetical order) are:

- chromIndex: an integer with the index of the chromatogram in the original source file (e.g. *mzML* file). In backedn with no original source file, this variable should be set to NA_integer_.
- collisionEnergy: for SRM data, numeric with the collision energy of the precursor.

- dataOrigin: optional character with the origin of a chromatogram.
- msLevel: integer defining the MS level of the data.
- mz: optional numeric with the (target) m/z value for the chromatographic data.
- mzMin: optional numeric with the lower m/z value of the m/z range in case the data (e.g. an extracted ion chromatogram EIC) was extracted from a Spectra object.
- mzMax: optional numeric with the upper m/z value of the m/z range.
- precursorMz: for SRM data, numeric with the target m/z of the precursor (parent).
- precursorMzMin: for SRM data, optional numeric with the lower m/z of the precursor's isolation window.
- precursorMzMax: for SRM data, optional numeric with the upper m/z of the precursor's isolation window.
- productMz for SRM data, numeric with the target m/z of the product ion.
- productMzMin: for SRM data, optional numeric with the lower m/z of the product's isolation window.
- productMzMax: for SRM data, optional numeric with the upper m/z of the product's isolation window.

Core Peaks variables

Similar to the *core* chromatogram variables, *core* peaks variables represent metadata that should be provided by a backend. Each of these variables should return a value, and if undefined, a missing value (with the appropriate data type) is returned. The number of values for a peaks variable in a single chromatogram can vary, from none to multiple, and may differ between chromatograms.

The names of peaks variables in the current chromatogram object can be obtained with the peaksVariables() function.

Each core peaks variable has a dedicated accessor method.

The corePeaksVariables() function returns the core peaks variables along with their expected (defined) data type.

The core peaks variables, listed in the required order for peaksData, are:

- rtime: A numeric vector containing retention time values.
- intensity: A numeric vector containing intensity values.

They should be provided for each chromatogram in the backend, **in this order**, No NAs are allowed for the rtime values. These characteristics will be checked with the validPeaksData() function.

Mandatory methods

New backend classes **must** extend the base ChromBackend class and implement the following mandatory methods:

- backendInitialize(): initialises the backend. This method is supposed to be called right after creating an instance of the backend class and should prepare the backend. Parameters can be defined freely for each backend, depending on what is needed to initialize the backend. This method has to ensure to set the chromatogram variable dataOrigin correctly.
- backendBpparam(): returns the parallel processing setup supported by the backend class. This function can be used by any higher level function to evaluate whether the provided parallel processing setup (or the default one returned by bpparam()) is supported by the backend. Backends not supporting parallel processing (e.g. because they contain a connection to a

database that can not be shared across processes) should extend this method to return only `SerialParam()` and hence disable parallel processing for (most) methods and functions. See also `backendParallelFactor()` for a function to provide a preferred splitting of the backend for parallel processing.

- `backendParallelFactor()`: returns a `factor` defining an optimal (preferred) way how the backend can be split for parallel processing used for all peak data accessor or data manipulation functions. The default implementation returns a factor of length 0 (`factor()`) providing thus no default splitting. `backendParallelFactor()` for `ChromBackendMzR` on the other hand returns `factor(dataOrigin(object))` hence suggesting to split the object by data file.
- `chromData()`, `chromData<-`: gets or sets general chromatogram metadata (annotation). `chromData()` returns a `data.frame`, `chromData<-` expects a `data.frame` with the same number of rows as there are chromatograms in `object`. Read-only backends might not need to implement the replacement method `chromData<-` (unless some internal caching mechanism could be used). `chromData()` should be implemented with the parameter `drop` set to `FALSE` as default. With `drop = FALSE` the method should return a `data.frame` even if one column is requested. If `drop = TRUE` is specified, the output will be a vector of the single column requested. New backends should be implemented such as if empty, the method returns a `data.frame` with 0 rows and the columns defined by `chromVariables()`. By default, the function *should* return at minimum the coreChromVariables, even if NAs.
- `chromExtract()`: return A new `Chrombackend` object containing separated chromatographic area as individual chromatograms. The chromatographic areas are defined by the `peak.table` parameter. The new object will contain chromatograms that match the conditions defined in `peak.table`. If no chromatograms match the conditions, an empty `ChromBackend` object should be returned.
- `extractByIndex()`: function to subset a backend to selected elements defined by the provided index. Similar to `[`, this method should allow extracting (or to subset) the data in any order. In contrast to `[`, however, `i` is expected to be an `integer` (while `[` should also support `logical` and eventually `character`). While being apparently redundant to `[`, this methods avoids package namespace errors/problems that can result in implementations of `[` being not found by R (which can happen sometimes in parallel processing using the `BiocParallel::SnowParam()`). This method is used internally to extract/subset its backend. Implementation of this method is mandatory.
- `peaksData()`: returns a `list` of `data.frame` with the data (e.g. retention time - intensity pairs) from each chromatogram. The length of the `list` is equal to the number of chromatograms in `object`. For an empty chromatogram a `data.frame` with 0 rows and two columns (named `"rtime"` and `"intensity"`) has to be returned. The optional parameter `columns`, if supported by the backend allows to define which peak variables should be returned in each array. As default (minimum) columns `"rtime"` and `"intensity"` have to be provided. `peaksData()` should be implemented with the parameter `drop` set to `FALSE` as default. With `drop = FALSE` the method should return a `data.frame` even if only one column is requested. If `drop = TRUE` is specified, the output will be a vector of the single column requested.
- `peaksData<-` replaces the peak data (retention time and intensity values) of the backend. This method expects a `list` of two-dimensional arrays (`data.frame`) with columns representing the peak variables. All existing peaks data are expected to be replaced with these new values. The length of the `list` has to match the number of chromatogram of `object`. Note that only writeable backends need to support this method.
- `[`: subset the backend. Only subsetting by element (*row*/`i`) is allowed. This method should be implemented as to support empty integer.
- `$`, `$<-`: access or set/add a single chromatogram variable (column) in the backend.

- backendMerge(): merges (combines) ChromBackend objects into a single instance. All objects to be merged have to be of the same type.

Optional methods with default implementations

Additional methods that might be implemented, but for which default implementations are already present are:

- [[
- backendParallelFactor(): returns a factor defining an optimal (preferred) way how the backend can be split for parallel processing used for all *peak* data accessor or data manipulation functions. The default implementation returns a factor of length 0 (factor()) providing thus no default splitting.
- chromIndex(): returns an integer vector with the index of the chromatograms in the original source file.
- chromVariables(): returns a character vector with the available chromatogram variables (columns, fields or attributes) available in object. Variables listed by this function are expected to be returned (if requested) by the chromData() function.
- collisionEnergy(), collisionEnergy<-: gets or sets the collision energy for the precursor (for SRM data). collisionEnergy() returns a numeric of length equal to the number of chromatograms in object.
- dataOrigin(), dataOrigin<-: gets or sets the *data origin* variable. dataOrigin() returns a character of length equal to the number of chromatograms, dataOrigin<- expects a character of length equal length(object).
- filterChromData(): filters any numerical chromatographic data variables based on the provided numerical ranges. The method should return a ChromBackend object with the chromatograms that match the condition. This function will results in an object with less chromatogram than the original.
- filterEmptyChromatograms(): removes empty chromatograms (i.e. chromatograms without peaks). Implementation of this method is optional since a default implementation for ChromBackend is available.
- intensity(): gets the intensity values from the chromatograms. Returns a list of numeric vectors (intensity values for each chromatogram). The length of the list is equal to the number of chromatograms in object.
- intensity<-: replaces the intensity values. value has to be a list of length equal to the number of chromatograms and the number of values within each list element identical to the number of data pairs in each chromatogram. Note that just writeable backends need to support this method.
- imputePeaksData(): Imputes missing intensity values in the chromatographic peaks data using various methods such as linear interpolation, spline interpolation, Gaussian kernel smoothing, or LOESS smoothing. This method modifies the peaks data in place and returns the same ChromBackend object with imputed values.
- isReadOnly(): returns a logical(1) whether the backend is *read only* or does allow also to write/update data. Defaults to FALSE.
- isEmpty(): returns a logical of length equal to the number of chromatograms with TRUE for chromatograms without any data pairs.
- length(): returns the number of chromatograms in the object.
- lengths(): returns the number of data pairs (retention time and intensity values) per chromatogram.

- `msLevel()`: gets the chromatogram's MS level. Returns an `integer` vector (of length equal to the number of chromatograms) with the MS level for each chromatogram (or `NA_integer_` if not available).
- `mz()`,`mz<-`: gets or sets the m/z value of the chromatograms. `mz()` returns a `numeric` of length equal to the number of chromatograms in `object`, `mz<-` expects a `numeric` of length `length(object)`.
- `mzMax()`,`mzMax<-`: gets or sets the upper m/z of the mass-to-charge range from which a chromatogram contains signal (e.g. if the chromatogram was extracted from MS data in spectra format and a m/z range was provided). `mzMax()` returns a `numeric` of length equal to the number of chromatograms in `object`, `mzMax<-` expects a `numeric` of length equal to the number of chromatograms in `object`.
- `mzMin()`,`mzMin<-`: gets or sets the lower m/z of the mass-to-charge range from which a chromatogram contains signal (e.g. if the chromatogram was extracted from MS data in spectra format and a m/z range was provided). `mzMin()` returns a `numeric` of length equal to the number of chromatograms in `object`, `mzMin<-` expects a `numeric` of length equal to the number of chromatograms in `object`.
- `peaksVariables()`: lists the available data variables for the chromatograms. Default peak variables are `"rtime"` and `"intensity"` (which all backends need to support and provide), but some backends might provide additional variables. Variables listed by this function are expected to be returned (if requested) by the `peaksData()` function.
- `precursorMz()`,`precursorMz<-`: gets or sets the (target) m/z of the precursor (for SRM data). `precursorMz()` returns a `numeric` of length equal to the number of chromatograms in `object`. `precursorMz<-` expects a `numeric` of length equal to the number of chromatograms.
- `precursorMzMin()`,`precursorMzMax()`,`productMzMin()`, `productMzMax()`: gets the lower and upper margin for the precursor or product isolation windows. These functions might return the value of `productMz()` if the respective minimal or maximal m/z values are not defined in `object`.
- `productMz()`,`productMz<-`: gets or sets the (target) m/z of the product (for SRM data). `productMz()` returns a `numeric` of length equal to the number of chromatograms in `object`. `productMz<-` expects a `numeric` of length equal to the number of chromatograms.
- `rtime()`: gets the retention times from the chromatograms. returns a `list` of `numeric` vectors (retention times for each chromatogram). The length of the returned list is equal to the number of chromatograms in `object`.
- `rtime<-`: replaces the retention times. `value` has to be a `list` of length equal to the number of chromatograms and the number of values within each list element identical to the number of data pairs in each chromatogram. Note that just writeable backends support this method.
- `split()`: splits the backend into a `list` of backends (depending on parameter `f`). The default method for `ChromBackend` uses `split.default()`, thus backends extending `ChromBackend` don't necessarily need to implement this method.
- `supportsSetBackend()`: whether a `ChromBackend` supports the `Chromatograms` `setBackend()` function. The default function will take the `peaksData()` and `chromData()` of the user's backend and pass it to the new backend. If the backend does not support this function, it should return `FALSE`. Therefore both backend in question should have a adequate `peaksData()` and `chromData()` method as well as their respective replacement method.

Implementation notes

Backends extending `ChromBackend` **must** implement all of its methods (listed above). A guide to create new backend classes is provided as a dedicated vignette. Additional information and an example for a backend implementation is provided in the respective vignette.

Author(s)

Johannes Rainer, Philippine Louail

Examples

```
## Create a simple backend implementation
ChromBackendDummy <- setClass("ChromBackendDummy",
    contains = "ChromBackend"
)

## We will show examples on a `ChromBackendMemory` backend.
be <- ChromBackendMemory()

## The `backendInitialize()` method initializes the backend filling it with
## data. This method can take any parameters needed for the backend to
## get loaded with the data.
cdata <- data.frame(
    msLevel = c(1L, 1L, 1L),
    mz = c(112.2, 123.3, 134.4),
    dataOrigin = c("mem1", "mem2", "mem3")
)

pdata <- list(
    data.frame(
        rtime = c(12.4, 12.8, 13.2, 14.6),
        intensity = c(123.3, 153.6, 2354.3, 243.4)
    ),
    data.frame(
        rtime = c(45.1, 46.2),
        intensity = c(100, 80.1)
    ),
    data.frame(
        rtime = c(12.4, 12.8, 13.2, 14.6),
        intensity = c(123.3, 153.6, 2354.3, 243.4)
    )
)
be <- backendInitialize(be, chromData = cdata, peaksData = pdata)

be

## Data can be accessed with the accessor methods
msLevel(be)

rtime(be)

## Even if no data was provided for all chromatogram variables, its accessor
## methods are supposed to return a value.
precursorMz(be)

## The `peaksData()` method is supposed to return data/frames of rtime and
## intensity pairs as a `list`.
peaksData(be)

## Use columns to extract specific peaks variables. Below we extract rtime
## and intensity values, but in reversed order to the default.
peaksData(be, columns = c("intensity", "rtime"))
```

```
## List available chromatographic variables
chromVariables(be)

## List available peak variables
peaksVariables(be)

## Extract multiple chromatographic variables
chromData(be, c("dataOrigin", "mz", "msLevel"))

## Single variables can also be accessed and replaced
mz(be)
mz(be) <- c(123.4, 134.5, 145.6)

be$msLevel
be$msLevel <- c(2L, 2L, 2L)

be[["rtime"]]
be[["rtime"]] <- list(
    c(12.4, 12.8, 13.2, 14.6),
    c(45.1, 46.2),
    c(12.4, 12.8, 13.2, 14.6)
)
```

`peaksData` *Chromatographic peaks data*

Description

As explained in the Chromatograms class documentation, the `Chromatograms` object is a container for chromatographic data that includes chromatographic peaks data (*retention time* and related intensity values, also referred to as *peaks data variables* in the context of `Chromatograms`) and metadata of individual chromatograms (so called *chromatograms variables*).

The *peaks data variables* information can be accessed using the `peaksData()` function. It is also possible to access specific peaks variables using `$`.

The peaks data can be accessed, replaced but also filtered/subsetted. Refer to the sections below for more details.

Usage

```
## S4 method for signature 'Chromatograms'
imputePeaksData(
  object,
  method = c("linear", "spline", "gaussian", "loess"),
  span = 0.3,
  sd = 1,
  window = 2,
  extrapolate = FALSE,
  ...
)

## S4 method for signature 'Chromatograms'
```

```
filterPeaksData(
  object,
  variables = character(),
  ranges = numeric(),
  match = c("any", "all"),
  keep = TRUE
)

## S4 method for signature 'Chromatograms'
intensity(object, ...)

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

## S4 method for signature 'Chromatograms'
peaksData(
  object,
  columns = peaksVariables(object),
  f = processingChunkFactor(object),
  BPPARAM = bpparam(),
  drop = FALSE,
  ...
)

## S4 replacement method for signature 'Chromatograms'
peaksData(object) <- value

## S4 method for signature 'Chromatograms'
peaksVariables(object, ...)

## S4 method for signature 'Chromatograms'
rtime(object, ...)

## S4 replacement method for signature 'Chromatograms'
rtime(object) <- value

## S4 method for signature 'Chromatograms'
lengths(x)

matchRtime(x, y, tolerance = Inf, ...)

## S4 method for signature 'Chromatograms,Chromatograms'
compareChromatograms(
  x,
  y,
  MAPFUN = matchRtime,
  FUN = cor,
  ...,
  minPeaks = 4L,
  BPPARAM = SerialParam()
)
```

```
## S4 method for signature 'Chromatograms,missing'
compareChromatograms(
  x,
  y = NULL,
  MAPFUN = matchRtime,
  FUN = cor,
  ...,
  minPeaks = 4L,
  labelsColumn = NULL,
  BPPARAM = SerialParam()
)

## S4 method for signature 'Chromatograms'
peakBoundary(
  object,
  threshold = 0.1,
  baselineThreshold = 0.1,
  baselineQuantile = 0.1,
  ...
)
```

Arguments

object	A Chromatograms object.
method	For imputePeaksData(): character(1): Imputation method ("linear", "spline", "gaussian", "loess").
span	For imputePeaksData: numeric(1), for the loess method: Smoothing parameter (only used if method == "loess")
sd	For imputePeaksData: numeric(1), for the gaussian method: Standard deviation for Gaussian kernel (only used if method == "gaussian")
window	For imputePeaksData: integer, for the gaussian method: Half-width of Gaussian kernel window (e.g., 2 gives window size 5)
extrapolate	For imputePeaksData: logical(1) (default FALSE). If TRUE, missing values at the beginning and end of a chromatogram (outside the range of observed values) will be extrapolated. If FALSE, only interpolation is performed and leading/trailing NA values remain NA.
...	Additional arguments passed to the method.
variables	For filterPeaksData(): character vector with the names of the peaks data variables to filter for. The list of available peaks data variables can be obtained with peaksVariables().
ranges	For filterPeaksData() : a numeric vector of paired values (upper and lower boundary) that define the ranges to filter the object. These paired values need to be in the same order as the variables parameter (see below).
match	For filterPeaksData() : character(1) defining whether the condition has to match for all provided ranges (match = "all"; the default), or for any of them (match = "any").
keep	For filterPeaksData(): logical(1) defining whether to keep (keep = TRUE) or remove (keep = FALSE) the chromatographic peaks data that match the condition.

value — For rtime() and intensity(): numeric vector with the values to replace the current values. The length of the vector must match the number of peaks data pairs in the Chromatograms object.

columns — For peaksData(): optional character with column names (peaks variables) that should be included in the returned list of data.frame. By default, all columns are returned. Available variables can be found by calling peaksVariables() on the object.

f — factor defining the grouping to split the Chromatograms object.

BPPARAM — Parallel setup configuration. See BiocParallel::bpparam() for more information.

drop — logical(1) For peaksData(), default to FALSE. If TRUE, and one column is called by the user, the method returns a list of vector of the single column requested.

x — For lengths() and compareChromatograms(): A Chromatograms object. For matchRtime(): a data.frame with columns rtime and intensity representing the first chromatogram.

y — For compareChromatograms(): A Chromatograms object against which x is compared. If missing, each chromatogram in x is compared with each other chromatogram in x. For matchRtime(): a data.frame with columns rtime and intensity representing the second chromatogram.

tolerance — For matchRtime(): numeric(1) (default Inf). Maximum RT difference between two measured points to be considered a match. Controls both the overlap detection and the shared RT grid. Lower values prevent a peak from being compared against a long interpolated gap in the other chromatogram. Use Inf (the default) to consider all RT points as matching. Can be forwarded via ... in compareChromatograms().

MAPFUN — For compareChromatograms(): function to align the retention times of two chromatograms before computing similarity. Must accept two data.frames (with columns rtime and intensity) and return a list with elements x and y: numeric vectors of equal length containing the aligned intensities of the first and second chromatogram respectively, interpolated onto a common retention-time grid. Defaults to matchRtime(). Additional arguments can be passed via ....

FUN — For compareChromatograms(): function to compute the similarity between two chromatograms from their aligned intensity vectors (as returned by MAPFUN). Must accept two numeric vectors as the first two arguments and return a single numeric value. Defaults to stats::cor() (Pearson correlation). Additional arguments can be passed via ... (e.g., method = "spearman" for stats::cor()).

minPeaks — For compareChromatograms(): integer(1) (default 4L). Minimum number of overlapping retention-time points (as returned by MAPFUN) required to compute a similarity score. Pairs whose retention-time overlap contains fewer than minPeaks points return NA in the score layer; the actual overlap count is still recorded in the n_peaks layer. Setting minPeaks = 2L recovers the behaviour of always computing a score whenever at least two points overlap.

labelsColumn — For compareChromatograms(): optional character(1) giving the name of a chromatogram variable (column in chromData()) whose values should be used as row and column names of the returned array. The column must contain unique values. If NULL (the default), the array dimensions are unnamed. Only used when y is missing.

`threshold` For `peakBoundary()`: `numeric(1)` (default `0.1`). Fraction of the peak height above baseline used as a fallback cut-off when valley-based boundaries are not suitable. Must be `>= 0` and `< 1`.

`baselineThreshold` For `peakBoundary()`: `numeric(1)` (default `0.1`). Fraction of the peak height above the baseline. Valley positions returned by `MsCoreUtils::valleys()` are accepted only if the intensity at the valley is at or below `baseline + peak_height * baselineThreshold`. Must be `>= 0` and `< 1`.

`baselineQuantile` For `peakBoundary()`: `numeric(1)` (default `0.1`). Quantile of the intensity distribution used as the baseline estimate. Must be `>= 0` and `<= 1`.

Value

Refer to the individual function description for information on the return value.

Filter Peaks Variables

Functions that filter a `Chromatograms`'s peaks data (i.e., `@peaksData`). These functions remove peaks data that do not meet the specified conditions. If a chromatogram in a `Chromatograms` object is filtered, only the corresponding peaks variable pairs (i.e., rows) in the `peaksData` are removed, while the chromatogram itself remains in the object.

The available functions to filter chromatographic peaks data include:

- `filterPeaksData()`: Filters numerical peaks data variables based on the specified numerical `ranges` parameter. This method returns the same input `Chromatograms` object, but the filtering step is added to the processing queue. The filtered data will be reflected when the user accesses `peaksData`. This function does *not* reduce the number of chromatograms in the object, but it removes the specified peaks data (e.g., "rtime" and "intensity" pairs) from the `peaksData`.

In the case of a read-only backend, (such as the ChromBackendMzR), the replacement of the peaks data is not possible. The peaks data can be filtered, but the filtered data will not be saved in the backend. This means the original mzML files will not be affected by computations performed on the Chromatograms.

Impute Peaks Variables

`imputePeaksData` will impute missing values in a `Chromatograms`'s peaks data (i.e., `@peaksData`). This functions replace missing peaks data values with specified imputation methods using various methods such as linear interpolation, spline interpolation, Gaussian kernel smoothing, or LOESS smoothing. This method modifies the peaks data in place and returns the same `Chromatograms` object with imputed values.

Peak Boundary Detection

`peakBoundary()` determines the retention time boundaries of the tallest peak in each chromatogram. The function uses `MsCoreUtils::valleys()` to locate the valleys (local minima) flanking the apex. If the valley intensities exceed a baseline-relative threshold (controlled by `baselineThreshold`), it falls back to a threshold-based boundary search using `threshold`. The baseline is estimated as the `baselineQuantile` quantile of the chromatogram's intensity values. The result is a `matrix` with one row per chromatogram and columns `left_boundary` and `right_boundary` (retention times). Chromatograms that are empty, have fewer than 3 data points, contain only `NA` or all-zero intensities return `NA` for both boundaries.

Compare Chromatograms

`compareChromatograms()` compares chromatograms in two steps:

1. **Align** – `MAPFUN` (default `matchRtime()`) maps two chromatograms onto a common retention-time grid and returns `list(x, y)`, where `x` and `y` are numeric vectors of equal length containing the aligned intensities of the first and second chromatogram respectively.
2. **Score** – `FUN` (default `stats::cor()`, Pearson correlation) computes a single similarity value from those aligned intensity vectors.

If `y` is missing, each chromatogram in `x` is compared against every other chromatogram in `x`; otherwise, each in `x` is compared with each in `y`.

The result is a 3-dimensional numeric array with dimensions `length(x) x length(y) x 2` (or symmetric `n x n x 2` for self-comparison). Layer `[, , 1]` (named `"score"`) contains pairwise similarity scores; layer `[, , 2]` (named `"n_peaks"`) contains the number of overlapping retention-time points used to compute each score. Pairs with fewer overlapping retention-time points than `minPeaks` (default 4) return `NA` in the score layer; the actual overlap count is still recorded in the `n_peaks` layer. The diagonal of a self-comparison is always 1 (score) and the number of data points in that chromatogram (count).

`matchRtime()` is the default `MAPFUN`. Given two chromatograms as `data.frames` with `rtime` and `intensity` columns, it aligns their RT axes and returns a named list with elements `x` and `y`: equal-length intensity vectors evaluated on a shared RT grid, ready for similarity scoring.

The alignment works as follows: `matchRtime()` first identifies the RT range where both chromatograms have measured points within `tolerance` of each other (the *overlap*). Within that range, it builds a shared RT grid from all of `x`'s RT points, adding any RT points from `y` that have no close match in `x` (within `tolerance`). Both intensity vectors are then linearly interpolated at grid positions they do not natively cover, using `stats::approx()`. If either chromatogram has fewer than 2 data points, or the two chromatograms do not overlap, empty vectors are returned.

The `tolerance` parameter (default `Inf`, meaning all RT points are considered matching) controls the strictness of the matching. Lowering it prevents comparing a measured peak against a long interpolated gap in the other chromatogram. Pass `tolerance` via `...` in `compareChromatograms()`.

When `y` is missing, the `labelsColumn` parameter assigns meaningful row/column names to the output from a `chromData()` column (e.g., `"mz"` or a user-defined feature identifier). The column must contain unique values. To compare groups of chromatograms separately, split the object with `split()` beforehand and apply `compareChromatograms()` to each subset.

Author(s)

Philippine Louail

See Also

Chromatograms for a general description of the `Chromatograms` object, and chromData for accessing,substituting and filtering chromatographic variables. For more information on the queuing of processings and parallelization for larger dataset processing see processingQueue.

Examples

```
# Create a Chromatograms object
cdata <- data.frame(
    msLevel = c(1L, 1L, 1L),
    mz = c(112.2, 123.3, 134.4),
    dataOrigin = c("mem1", "mem2", "mem3")
```

```
)

pdata <- list(
    data.frame(
        rtime = c(12.4, 12.8, 13.2, 14.6),
        intensity = c(123.3, 153.6, 2354.3, 243.4)
    ),
    data.frame(
        rtime = c(45.1, 46.2),
        intensity = c(100, 80.1)
    ),
    data.frame(
        rtime = c(12.4, 12.8, 13.2, 14.6),
        intensity = c(123.3, 153.6, 2354.3, 243.4)
    )
)

be <- backendInitialize(new("ChromBackendMemory"),
    chromData = cdata,
    peaksData = pdata
)

chr <- Chromatograms(be)

# Access peaks data
peaksData(chr)

# Access specific peaks data variables
peaksData(chr, columns = "rtime")
rtime(chr)

# Replace peaks data
rtime(chr)[[1]] <- c(1, 2, 3, 4)

# Filter peaks data
filterPeaksData(chr, variables = "rtime", ranges = c(12.5, 13.5))

# Pairwise similarity: returns a 3D array [i, j, layer]
res <- compareChromatograms(chr)
res[, , "score"]   ## similarity scores
res[, , "n_peaks"] ## number of overlapping RT points

## Use Spearman correlation (passed to cor() via ...)
compareChromatograms(chr, method = "spearman")[, , "score"]

# Use a chromData column as row/column labels
compareChromatograms(chr, labelsColumn = "mz")[, , "score"]

# Compare two Chromatograms objects
compareChromatograms(chr[1:2], chr[3])
```

`plotChromatograms` *Plot chromatograms*

Description

Chromatograms() can be plotted with the following functions:

The plotChromatograms(): plots each chromatogram in its separate plot by splitting the plot area into as many panels as there are spectra.

Usage

```
plotChromatograms(
  x,
  xlab = "rtime (s)",
  ylab = "intensity",
  type = "o",
  pch = 20,
  cex = 0.6,
  lwd = 1.5,
  xlim = numeric(),
  ylim = numeric(),
  main = character(),
  col = "#00000080",
  asp = 1,
  ...
)

plotChromatogramsOverlay(
  x,
  xlab = "rtime (s)",
  ylab = "intensity",
  type = "o",
  pch = 20,
  cex = 0.6,
  lwd = 1.5,
  xlim = numeric(),
  ylim = numeric(),
  main = paste(length(x), "chromatograms"),
  col = "#00000080",
  axes = TRUE,
  frame.plot = axes,
  ...
)
```

Arguments

x	A Chromatograms object.
xlab	character(1) with the label for the x-axis (by default xlab = "rtime (s)").
ylab	character(1) with the label for the y-axis (by default ylab = "intensity").
type	character(1) specifying the type of plot. See plot.default() for details. Defaults to type = "l" which draws each peak as a line.
pch	integer(1) or character(1) specifying the plotting symbol (see plot.default()).
cex	numeric(1) specifying the size of the plotting symbol (see plot.default()).
lwd	numeric(1) specifying the line width (see plot.default()).

xlim	numeric(2) defining the x-axis limits. The range of m/z values are used by default.
ylim	numeric(2) defining the y-axis limits. The range of intensity values are used by default.
main	character(1) with the title for the plot. By default the spectrum's MS level and retention time (in seconds) is used.
col	color to be used to draw the peaks. Should be either of length 1, or equal to the number of chromatograms (to plot each chromatograms in a different color) or be a list with colors for each individual peak in each spectrum.
asp	numeric(1) the aspect ratio of the plot, i.e. the ratio of the y-axis to the x-axis. Defaults to 1.
...	Additional arguments to be passed to plot.default().
axes	logical(1) whether (x and y) axes should be drawn.
frame.plot	logical(1) whether a box should be drawn around the plotting area.

Value

These functions create a plot.

Refer to the individual function description for information on the return value.

Author(s)

Philippine Louail, Johannes Rainer.

Examples

```
## Create a Chromatograms object
cdata <- data.frame(
    msLevel = c(1L, 1L, 1L),
    mz = c(112.2, 123.3, 134.4),
    chromIndex = c(1L, 2L, 3L)
)
pdata <- list(
    data.frame(
        rtime = c(12.4, 12.8, 13.2, 14.6),
        intensity = c(123.3, 153.6, 2354.3, 243.4)
    ),
    data.frame(
        rtime = c(45.1, 46.2),
        intensity = c(100, 80.1)
    ),
    data.frame(
        rtime = c(12.4, 12.8, 13.2, 14.6),
        intensity = c(123.3, 153.6, 2354.3, 243.4)
    )
)
chr <- backendInitialize(ChromBackendMemory(),
    chromData = cdata,
    peaksData = pdata
) |> Chromatograms()

## Plot one chromatogram
plotChromatograms(chr[1])
```

```
## Plot the full Chromatograms object
plotChromatograms(chr)

## Define a color for each peak in each chromatogram
plotChromatograms(chr[1:2], col = c("green", "blue"))

## Overlay all chromatograms
plotChromatogramsOverlay(chr[1:2], col = c("green", "blue"))
```

---

processingQueue *Efficiently processing* Chromatograms *objects.*

Description

The processingQueue of a Chromatograms object is a list of processing steps (i.e., functions) that are stored within the object and applied only when needed. This design allows data to be processed in a single step, which is particularly useful for larger datasets. The processing queue enables functions to be applied in a chunk-wise manner, facilitating parallel processing and reducing memory demand.

Since the peaks data can be quite large, a processing queue is used to ensure efficiency. Generally, the processing queue is applied either temporarily when calling peaksData() or permanently when calling applyProcessing(). As explained below the processing efficiency can be further improved by enabling chunk-wise processing.

Usage

```
## S4 method for signature 'Chromatograms'
applyProcessing(
  object,
  f = processingChunkFactor(object),
  BPPARAM = bpparam(),
  ...
)

## S4 method for signature 'Chromatograms'
addProcessing(object, FUN, ...)

## S4 method for signature 'Chromatograms'
processingChunkSize(object, ...)

## S4 replacement method for signature 'Chromatograms'
processingChunkSize(object) <- value

## S4 method for signature 'Chromatograms'
processingChunkFactor(object, chunkSize = processingChunkSize(object), ...)
```

Arguments

object	A Chromatograms object.

f	factor defining the grouping to split the Chromatograms object.
BPPARAM	Parallel setup configuration. See BiocParallel::bpparam() for more information.
...	Additional arguments passed to the methods.
FUN	For addProcessing(), a function to be added to the Chromatograms object's processing queue.
value	integer(1) defining the chunk size.
chunkSize	integer(1) for processingChunkFactor defining the chunk size. The default is the value stored in the Chromatograms object's processingChunkSize slot.

Value

processingChunkSize() returns the currently defined processing chunk size (or Inf if it is not defined). processingChunkFactor() returns a factor defining the chunks into which object will be split for (parallel) chunk-wise processing or a factor of length 0 if no splitting is defined.

Refer to the individual function description for information on the return value.

Apply Processing

The applyProcessing() function applies the processing queue to the backend and returns the updated Chromatograms object. The processing queue is a list of processing steps applied to the chromatograms data. Each element in the list is a function that processes the chromatograms data. To apply processing to the peaks data, the backend must be set to a non-read-only backend using the setBackend() function.

Parallel and Chunk-wise Processing of Chromatograms

Many operations on Chromatograms objects, especially those involving the actual peaks data (see peaksData), support chunk-wise processing. This involves splitting the Chromatograms into smaller parts (chunks) that are processed iteratively. This enables parallel processing by data chunk and reduces memory demand since only the peak data of the currently processed subset is loaded into memory. Chunk-wise processing, which is disabled by default, can be enabled by setting the processing chunk size of a Chromatograms object using the processingChunkSize() function to a value smaller than the length of the Chromatograms object. For example, setting processingChunkSize(chr) <- 1000 will cause any data manipulation operation on chr, such as filterPeaksData(), to be performed in parallel for sets of 1000 chromatograms in each iteration.

Chunk-wise processing is particularly useful for Chromatograms objects using an *on-disk* backend or for very large experiments. For small datasets or Chromatograms using an in-memory backend, direct processing might be more efficient. Setting the chunk size to Inf will disable chunk-wise processing.

Some backends may prefer a specific type of splitting and chunk-wise processing. For example, the ChromBackendMzR backend needs to load MS data from the original (mzML) files, so chunk-wise processing on a per-file basis is ideal. The backendParallelFactor() function for ChromBackend allows backends to suggest a preferred data chunking by returning a factor defining the respective data chunks. The ChromBackendMzR returns a factor based on the *dataOrigin* chromatograms variable. A factor of length 0 is returned if no particular preferred splitting is needed. The suggested chunk definition will be used if no finite processingChunkSize() is defined. Setting the processingChunkSize overrides backendParallelFactor.

Functions to configure parallel or chunk-wise processing:

- `processingChunkSize()`: Gets or sets the size of the chunks for parallel or chunk-wise processing of a `Chromatograms` object. With a value of `Inf` (the default), no chunk-wise processing will be performed.
- `processingChunkFactor()`: Returns a `factor` defining the chunks into which a `Chromatograms` object will be split for chunk-wise (parallel) processing. A `factor` of length 0 indicates that no chunk-wise processing will be performed.

Note

Some backends might not support parallel processing. For these, the `backendBpparam()` function will always return a `SerialParam()` regardless of how parallel processing was defined.

Author(s)

Johannes Rainer, Philippine Louail

Examples

```
# Create a Chromatograms object
cdata <- data.frame(
    msLevel = c(1L, 1L, 1L),
    mz = c(112.2, 123.3, 134.4),
    chromIndex = c(1L, 2L, 3L)
)

pdata <- list(
    data.frame(
        rtime = c(12.4, 12.8, 13.2, 14.6),
        intensity = c(123.3, 153.6, 2354.3, 243.4)
    ),
    data.frame(
        rtime = c(45.1, 46.2),
        intensity = c(100, 80.1)
    ),
    data.frame(
        rtime = c(12.4, 12.8, 13.2, 14.6),
        intensity = c(123.3, 153.6, 2354.3, 243.4)
    )
)

be <- backendInitialize(new("ChromBackendMemory"),
    chromData = cdata,
    peaksData = pdata
)

chr <- Chromatograms(be)

divide_intensities <- function(x, y, ...) {
    intensity(x) <- lapply(intensity(x), `/`, y)
    x
}

## Add the function to the procesing queue
chr <- addProcessing(chr, divide_intensities, y = 2)
chr
```

```
# Apply the processing queue
chr <- applyProcessing(chr)
```

`reset` *Fill data.frame with columns for missing core chromatogram variables.*

Description

`fillCoreChromVariables()` fills a provided `data.frame` with columns for eventually missing *core* chromatogram variables. The missing core variables are added as new columns with missing values (`NA`) of the correct data type. Use `coreChromVariables()` to list the set of core variables and their data types.

`validChromData()` checks that columns, representing *core* chromatogram variables are of the correct data type.

For S4 methods that require a documentation entry but only clutter the index.

This method returns the chromatographic data stored in the backend. If not specified otherwise it will return all defined columns in the chromData slot as well as adding the `coreChromVariables` missing with NA values.

Usage

```
reset(object, ...)

fillCoreChromVariables(x = data.frame())

validChromData(x = data.frame(), error = TRUE)

validPeaksData(x = list(), error = TRUE)

## S4 method for signature 'ChromBackendMemory'
backendMerge(object, ...)

## S4 method for signature 'ChromBackendMemory'
chromData(object, columns = chromVariables(object), drop = FALSE)

## S4 replacement method for signature 'ChromBackendMemory'
chromData(object) <- value

## S4 method for signature 'ChromBackendMemory'
chromVariables(object)

## S4 method for signature 'ChromBackendMemory'
peaksData(object, columns = peaksVariables(object), drop = FALSE, ...)

## S4 replacement method for signature 'ChromBackendMemory'
peaksData(object) <- value

## S4 method for signature 'ChromBackendMemory'
```

```
peaksVariables(object)

## S4 method for signature 'ChromBackendMemory'
backendParallelFactor(object, ...)

## S4 method for signature 'ChromBackendMemory'
isReadOnly(object)

## S4 method for signature 'ChromBackendMemory'
lengths(x)

## S4 method for signature 'ChromBackendMemory'
intensity(object)

## S4 method for signature 'ChromBackendMemory'
rtime(object)

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

## S4 method for signature 'ChromBackendMemory'
supportsSetBackend(object, ...)

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

## S4 method for signature 'ChromBackendMemory'
x$name

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

## S4 method for signature 'ChromBackendMemory'
chromExtract(object, peak.table, by)

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

## S4 method for signature 'ChromBackendMzR'
backendParallelFactor(object, ...)

## S4 method for signature 'ChromBackendMzR'
isReadOnly(object)

## S4 method for signature 'ChromBackendMzR'
peaksData(
  object,
  columns = peaksVariables(object),
  drop = FALSE,
  BPPARAM = SerialParam(),
  ...
)
```

```
## S4 replacement method for signature 'ChromBackendMzR'
peaksData(object) <- value

## S4 replacement method for signature 'ChromBackendMzR'
chromData(object) <- value

## S4 method for signature 'ChromBackendMzR'
supportsSetBackend(object, ...)

## S4 method for signature 'ChromBackendMzR'
intensity(object)

## S4 method for signature 'ChromBackendMzR'
rtime(object)

## S4 method for signature 'ChromBackendMzR'
lengths(x)

## S4 method for signature 'ChromBackendMzR'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'ChromBackendMzR'
chromExtract(object, peak.table, by, ...)

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

## S4 method for signature 'ChromBackendSpectra'
factorize(object, factorize.by = c("msLevel", "dataOrigin"), ...)

## S4 method for signature 'ChromBackendSpectra'
backendParallelFactor(object, ...)

## S4 method for signature 'ChromBackendSpectra'
isReadOnly(object)

## S4 method for signature 'ChromBackendSpectra'
peaksData(object, columns = peaksVariables(object), drop = FALSE, ...)

## S4 replacement method for signature 'ChromBackendSpectra'
peaksData(object) <- value

## S4 method for signature 'ChromBackendSpectra'
supportsSetBackend(object, ...)

## S4 method for signature 'ChromBackendSpectra'
intensity(object)

## S4 method for signature 'ChromBackendSpectra'
rtime(object)
```

```
## S4 method for signature 'ChromBackendSpectra'
lengths(x)

## S4 method for signature 'ChromBackendSpectra'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'ChromBackendSpectra'
chromExtract(object, peak.table, by, ...)

## S4 method for signature 'Chromatograms'
show(object)
```

Arguments

object	A Chromatograms object.
x	list representing the peaks data of a Chromatograms
error	logical(1) whether an error should be thrown (the default) if one or more columns don't have the correct data type.

Value

input data frame x with missing core variables added (with the correct data type).

If the core variables have all the correct data type: an empty character. If one or more core variables (columns) have the wrong data type the function either throws an error (with error = TRUE) or returns a character specifying which variables/columns don't have the correct type (for error = FALSE).

Not applicable

Examples

```
## Define a data frame
a <- data.frame(msLevel = c(1L, 1L, 2L), other_column = "b")

## Add missing core chromatogram variables to this data frame
fillCoreChromVariables(a)

## The data.frame thus contains columns for all core chromatogram
## variables in the respective expected data type (but filled with
## missing values).
```

Index