# Package: Chromatograms (via r-universe)

September 2, 2024

**Title** Infrastructure for Chromatographic Mass Spectrometry Data

**Version** 0.1.0

**Description** The Chromatograms packages defines a 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** ProtGenerics (>= 1.35.4)

**Imports** methods, S4Vectors, MsCoreUtils (>= 1.7.5), IRanges

**Suggests** testthat, knitr (>= 1.1.0), rmarkdown, BiocStyle

**License** Artistic-2.0

**Encoding** UTF-8

**LazyData** yes

**VignetteBuilder** knitr

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

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

**biocViews** Infrastructure, Proteomics, MassSpectrometry, Metabolomics

**Roxygen** list(markdown=TRUE)

**RoxygenNote** 7.3.2

**Collate** 'AllGenerics.R' 'ChromBackend-functions.R' 'ChromBackend.R'

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

**RemoteUrl** https://github.com/rformassspectrometry/Chromatograms

**RemoteRef** HEAD

**RemoteSha** 68a9966c2ffd5ca6f99f42152cedd4141aa6acd5

## Contents

chromData . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
fillCoreChromVariables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
validChromData . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
validPeaksData . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**Index** **14**

## chromData *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 implemet 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.

### Usage

```
chromData(object, ...)

chromData(object) <- value

chromIndex(object, ...)

chromIndex(object) <- value

chromNames(object, ...)

chromNames(object) <- value

chromVariables(object, ...)

mzMax(object, ...)

mzMax(object) <- value

mzMin(object, ...)
mzMin(object) <- value

precursorMzMin(object, ...)

precursorMzMin(object) <- value

precursorMzMax(object, ...)

precursorMzMax(object) <- value

productMzMax(object, ...)

productMzMax(object) <- value

productMzMin(object, ...)

productMzMin(object) <- value

selectChromVariables(object, ...)

reset(object, ...)

coreChromVariables()

corePeaksVariables()

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

## 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'
peaksData(object, columns = c("rtime", "intensity"), drop = FALSE)

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

```
peaksData(object) <- value

## S4 method for signature 'ChromBackend'
selectChromVariables(object, chromVariables = chromVariables(object))

## 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'
chromVariables(object)

## S4 method for signature 'ChromBackend'
chromIndex(object, columns = chromVariables(object))

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

## 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'
dataStorage(object)

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

## 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, ...)
```

## Arguments

| | |
|---|---|
| `object` | Object extending `ChromBackend`. |
| `...` | Additional arguments. |
| `value` | replacement value for `<-` methods. See individual method description or expected data type. |
| `x` | Object extending `ChromBackend`. |
| `i` | For `[`: `integer`, `logical` or `character` to subset the object. |
| `j` | For `[`: ignored. |

| | |
|---|---|
| `drop` | For `chromData()` and `peaksData()`: `logical(1)` default to `FALSE`. If `TRUE`, and one column is called by the user, the method should return a vector (or list of vector for `peaksData()`) of the single column requested. |
| `name` | For `$` and `$<-`: the name of the chromatogram variable to return or set. |
| `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. |
| `chromVariables` | For `selectChromVariables()`: `character` with the names of the chromatogram variables to which the backend should be subsetted. |
| `f` | `factor` defining the grouping to split `x`. See `split()`. |
| `dataOrigin` | For `filterDataOrigin()`: `character` to define which chromatograms to keep. |
| `dataStorage` | For `filterDataStorage()`: `character` to define which chromatograms to keep. |
| `msLevel` | `integer` defining the MS level of the chromatograms to which the function should be applied. For `filterMsLevel()`: the MS level to which `object` should be subsetted. |
| `mz` | For `filterMzValues()`: `numeric` with the m/z values of chromatograms to keep. All chromatograms with their `mz` chromatogram variable matching any of the values provided with this parameter are retained. Parameters `ppm` and `tolerance` allow relaxed matching. For `filterMzRange()`: `numeric(2)` defining the lower and upper boundary of the m/z range. Chromatograms with their `mz` chromatogram variable within this range are retained. |
| `ppm` | For `filterMzValues()`: m/z-relative acceptable difference (in parts-per-million) for m/z values to be considered *matching*. |
| `tolerance` | For `filterMzValues()`: largest acceptable absolute difference in m/z values to consider them *matching*. |

## 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 core variables are returned with the `chromVariables` function.

For each core chromatogram variable a dedicated access method exists.

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).
- `collisionEnergy`: for SRM data, `numeric` with the collision energy of the precursor.
- `dataOrigin`: optional `character` with the origin of a chromatogram.
- `dataStorage`: `character` defining where the data is (currently) stored.
- `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.

## Backend functions

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 spectra variable `dataStorage` correctly.
- `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 only one column is called. If `drop = TRUE` is specified, the output will be a vector of the single column requested.
- `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 called. 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 spectra of `object`. Note that only writeable backends need to support this method.
- `[`: subset the backend. Only subsetting by element (*row*/`i`) is allowed.
- `$`, `$<-`: access or set/add a single chromatogram variable (column) in the backend.

- `selectChromVariables()`: reduce `object` retaining only specified chromatogram variables.
- `backendMerge()`: merges (combines) `ChromBackend` objects into a single instance. All objects to be merged have to be of the same type.

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.
- `chromVariables()`: returns a `character` vector with the available chromatogram variables (columns, fields or attributes) available in `object`.
- `chromIndex()`: returns an `integer` vector with the index of the chromatograms in the original source file.
- `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)`.
- `dataStorage()`, `dataStorage<-`: gets or sets the *data storage* variable. `dataStorage()` returns a `character` of length equal to the number of chromatograms in `object`, `dataStorage<-` expects a `character` of length equal `length(object)`. Note that missing values (`NA_character_`) are not supported for `dataStorage()`.
- `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.
- `isReadOnly()`: returns a `logical(1)` whether the backend is *read only* or does allow also to write/update data.
- `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 `NumericList()` 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` (or `NumericList()`) 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.

Filter methods:

- `filterDataOrigin()`: filters the object retaining chromatograms matching any of the provided `dataOrigin`. Parameter `dataOrigin` has to be of type `character` and needs to match exactly the data origin value of the chromatograms to subset. `filterDataOrigin()` should return the data ordered by the provided `dataOrigin` parameter, i.e. if `dataOrigin = c("2", "1")` was provided, the chromatograms in the resulting object should be ordered accordingly (first chromatogram from data origin `"2"` and then from `"1"`).
- `filterDataStorage()`: filters the object retaining chromatograms matching any of the provided `dataStorage`. Parameter `dataStorage` has to be of type `character` and needs to match exactly the data storage value of the chromatograms to subset. `filterDataStorage()` should return the data ordered by the provided `dataStorage` parameter, i.e. if `dataStorage`

= `c("2", "1")` was provided, the chromatograms in the resulting object should be ordered accordingly (first chromatogram from data storage `"2"` and then from `"1"`).

- `filterMsLevel()`: retains chromatograms of MS level `msLevel()`.
- `filterMzRange()`: retains chromatograms with their m/z within the provided m/z range.
- `filterMzValues()`: retains chromatograms with their m/z matching any of the provided m/z values (given the provided acceptable differences defined by parameters `tolerance` and `ppm`.

## 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")
```

---

`fillCoreChromVariables` *Fill data frame with columns for missing core chrom 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.

## Usage

```
fillCoreChromVariables(x = data.frame())
```

## Arguments

| | |
|---|---|
| `x` | `data.frame` with potentially present core chrom variable columns |

## Value

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

## 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).
```

---

`validChromData` *Check core chromatogram variables for correct data types*

---

## Description

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

## Usage

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

## Arguments

| | |
|---|---|
| x | `data.frame` representing metadata 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

If 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`).

---

`validPeaksData` `validPeaksData()` *checks that the names of the input peaksData list, representing* core *peaks variables are of the correct data type.*

---

## Description

`validPeaksData()` checks that the names of the input peaksData list, representing *core* peaks variables are of the correct data type.

## Usage

```
validPeaksData(x = list())
```

## Arguments

| | |
|---|---|
| `x` | `list` representing the peaks data of a `Chromatograms` |

# Index

[,ChromBackend-method *(chromData)*, 2
$,ChromBackend-method *(chromData)*, 2
$<-,ChromBackend-method *(chromData)*, 2

backendInitialize,ChromBackend-method *(chromData)*, 2
backendMerge,ChromBackend-method *(chromData)*, 2
backendMerge,list-method *(chromData)*, 2
backendParallelFactor,ChromBackend-method *(chromData)*, 2

ChromBackend *(chromData)*, 2
ChromBackend-class *(chromData)*, 2
chromData, 2
chromData,ChromBackend-method *(chromData)*, 2
chromData<- *(chromData)*, 2
chromData<-,ChromBackend-method *(chromData)*, 2
chromIndex *(chromData)*, 2
chromIndex,ChromBackend-method *(chromData)*, 2
chromIndex<- *(chromData)*, 2
chromIndex<-,ChromBackend-method *(chromData)*, 2
chromNames *(chromData)*, 2
chromNames<- *(chromData)*, 2
chromVariables *(chromData)*, 2
chromVariables,ChromBackend-method *(chromData)*, 2
collisionEnergy,ChromBackend-method *(chromData)*, 2
collisionEnergy<-,ChromBackend-method *(chromData)*, 2
coreChromVariables *(chromData)*, 2
coreChromVariables(), *11*
corePeaksVariables *(chromData)*, 2

dataOrigin,ChromBackend-method *(chromData)*, 2
dataOrigin<-,ChromBackend-method *(chromData)*, 2
dataStorage,ChromBackend-method *(chromData)*, 2
dataStorage<-,ChromBackend-method *(chromData)*, 2

fillCoreChromVariables, 11

intensity,ChromBackend-method *(chromData)*, 2
intensity<-,ChromBackend-method *(chromData)*, 2
isEmpty,ChromBackend-method *(chromData)*, 2
isReadOnly,ChromBackend-method *(chromData)*, 2

length,ChromBackend-method *(chromData)*, 2
lengths,ChromBackend-method *(chromData)*, 2

msLevel,ChromBackend-method *(chromData)*, 2
msLevel<-,ChromBackend-method *(chromData)*, 2
mz,ChromBackend-method *(chromData)*, 2
mz<-,ChromBackend-method *(chromData)*, 2
mzMax *(chromData)*, 2
mzMax,ChromBackend-method *(chromData)*, 2
mzMax<- *(chromData)*, 2
mzMax<-,ChromBackend-method *(chromData)*, 2
mzMin *(chromData)*, 2
mzMin,ChromBackend-method *(chromData)*, 2
mzMin<- *(chromData)*, 2
mzMin<-,ChromBackend-method *(chromData)*, 2

NumericList(), *10*

peaksData,ChromBackend-method
(chromData), 2
peaksData<-,ChromBackend-method
(chromData), 2
peaksVariables,ChromBackend-method
(chromData), 2
precursorMz,ChromBackend-method
(chromData), 2
precursorMz<-,ChromBackend-method
(chromData), 2
precursorMzMax (chromData), 2
precursorMzMax,ChromBackend-method
(chromData), 2
precursorMzMax<- (chromData), 2
precursorMzMax<-,ChromBackend-method
(chromData), 2
precursorMzMin (chromData), 2
precursorMzMin,ChromBackend-method
(chromData), 2
precursorMzMin<- (chromData), 2
precursorMzMin<-,ChromBackend-method
(chromData), 2
productMz,ChromBackend-method
(chromData), 2
productMz<-,ChromBackend-method
(chromData), 2
productMzMax (chromData), 2
productMzMax,ChromBackend-method
(chromData), 2
productMzMax<- (chromData), 2
productMzMax<-,ChromBackend-method
(chromData), 2
productMzMin (chromData), 2
productMzMin,ChromBackend-method
(chromData), 2
productMzMin<- (chromData), 2
productMzMin<-,ChromBackend-method
(chromData), 2

reset (chromData), 2
reset,ChromBackend-method (chromData), 2
rtime,ChromBackend-method (chromData), 2
rtime<-,ChromBackend-method
(chromData), 2

selectChromVariables (chromData), 2
selectChromVariables,ChromBackend-method
(chromData), 2
split(), *7*
split,ChromBackend,ANY-method
(chromData), 2
split.default(), *10*

validChromData, 12
validPeaksData, 12