Package ‘TargetSearch’

February 28, 2026

Type Package

Title A package for the analysis of GC-MS metabolite profiling data

Version 2.12.0

Date 2025-08-04

Maintainer Alvaro Cuadros-Inostroza <acuadros+bioc@gmail.com>

Imports graphics, grDevices, methods, ncdf4, stats, utils, assertthat

Suggests TargetSearchData, BiocStyle, knitr, tinytest

VignetteBuilder knitr

Description This packages provides a flexible, fast and accurate method
for targeted pre-processing of GC-MS data. The user provides a (often
very large) set of GC chromatograms and a metabolite library of targets.
The package will automatically search those targets in the chromatograms
resulting in a data matrix that can be used for further data analysis.

biocViews MassSpectrometry, Preprocessing, DecisionTree,
ImmunoOncology

License GPL (>= 2)

URL https://github.com/acinostroza/TargetSearch

BugReports https://github.com/acinostroza/TargetSearch/issues

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

git_branch RELEASE_3_22

git_last_commit ab9658a

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2026-02-27

Author Alvaro Cuadros-Inostroza [aut, cre],
Jan Lisec [aut],
Henning Redestig [aut],
Matt Hannah [aut]

Contents

`TargetSearch-package` *A targeted approach for GC-MS data.*

Description

This packages provides a targeted method for GC-MS data analysis. The workflow includes a peak picking algorithm to convert from netcdf files to tab delimited files, retention time correction using retention time markers provided by the user, and a library search using multiple marker masses and retention time index optimisation.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

Maintainer: Alvaro Cuadros-Inostroza <inostroza@mpimp-golm.mpg.de>

`baseline` *Baseline correction - wrapper function*

Description

This function perform baseline correction by wrapping around the methods implemented on `baselineCorrection` and `baselineCorrectionQuant`.

Usage

```
baseline(ncdf, bsline_method = c('classic', 'quantiles', 'none'), ...)
```

Arguments

`ncdf`	A list containing the raw chromatogram data. The list can be generated by `peakCDFextraction`. Expected elements are `"Peaks"` which is matrix of intensities where the rows are retention times and columns are mass traces, `"Time"` which is a vector of retention time in seconds.
`bsline_method`	A string to select the baseline retention method. Options are `"classic"` which implements Chang's method (the old or classic TargetSearch method), `"quantiles"` the quantiles based method, and `"none"` which does nothing (returns the same input).
`...`	Extra parameters to be passed to `baselineCorrection` or `baselineCorrectionQuant`

.

Details

This is a wrapper function around the different baseline correction algorithms. It is not intended to be executed by the average user. Please refer to the respective man pages for details.

Value

Returns a list with the same elements as the input, but the element "Peaks" containing baseline corrected values.

Author(s)

Alvaro Cuadros-Inostroza

See Also

RIcorrect, baselineCorrection, baselineCorrectionQuant

Examples

```
# get a random sample CDF from TargetSearchData
require(TargetSearchData)
cdffile <- sample(tsd_cdffiles(), 1)
pdata <- peakCDFextraction(cdffile)

# restrict mass range to reduce computing time (not needed for
# actual data)
pdata$Peaks <- pdata$Peaks[, 1:10] ; pdata$massRange <- c(85, 94)

# make a fake baseline as constant + noise (the CDF files have been
# already baseline corrected by the vendor software).
nscans <- length(pdata$Time)
noise <- as.integer(1000 + rnorm(nscans, sd=5))
pdata$Peaks <- pdata$Peaks + noise

# use Classic and Quantile methods for baseline correction (def parameters)
pdata_c <- baseline(pdata, 'classic')
# use Quantile method
pdata_q <- baseline(pdata, 'quantiles')

# plot function to compare traces
plfun <- function(p, q, k, n, titl) {
  plot(p$Time, p$Peaks[, k], col='blue', type='l', xlab='time', ylab='intensity')
  lines(q$Time, q$Peaks[, k] - n, col='red')
  legend('topleft', c('corrected', 'original'), col=c('blue', 'red'), lty=1, lwd=2)
  title(paste('method:', titl))
}

op <- par(mfrow=c(2,2))
  plfun(pdata_c, pdata, 1, noise, 'classic')
  plfun(pdata_q, pdata, 1, noise, 'quantile')
  plfun(pdata_c, pdata, 7, noise, 'classic')
  plfun(pdata_q, pdata, 7, noise, 'quantile')
par(op)
```

`baselineCorrection` *Baseline correction - Chang's method*

Description

Function for baseline correction of GC-MS chromatograms using Chang's method described below.

Usage

```
baselineCorrection(peaks, threshold = 0.5, alpha = 0.95, bfraction = 0.2,
    segments = 100, signalWindow = 10, method = "linear")
```

Arguments

`peaks`	Either a matrix object of spectra peak intensities to be baseline corrected, where the rows are retention times and columns are mass traces; or, a named list containing an element called `"Peaks"` which such matrix. The list can be generated by `peakCDFextraction`
`threshold`	A numeric value between 0 and 1. A value of one sets the baseline above the noise, 0.5 in the middle of the noise and 0 below the noise.
`alpha`	The alpha parameter of the high pass filter.
`bfraction`	The percentage of the fragments with the lowest intensities of the filtered signal that are assumed to be baseline signal.
`segments`	The number of segments in which the filtered signal is divided.
`signalWindow`	The window size (number of points) used in the signal windowing step.
`method`	The method used to approximate the baseline. `"linear"` (default) uses linear interpolation. `"spline"` fits a cubic smoothing spline (warning: really slow).

Details

The baseline correction algorithm is based on the work of Chang et al, and it works as follows. For every mass trace, i.e., columns of matrix `peaks`, the signal intensity is filtered by a first high pass filter:

$$y_i = \alpha(y_{i-1} + x_i - x_{i-1})$$

The filtered signal is divided into evenly spaced segments (`segments`) and the standard deviation of each segment is calculated. A percentage (`bfraction`) of the segments with the lowest values are assumed to be baseline signal and the standard deviation (σ) of the points within those segments is calculated.

Once σ has been determined, the points with absolute filtered values larger than 2σ are considered signal. After that, the signal windowing step takes every one of the points found to be signal as the center of a signal window (`signalWindow`) and marks the points within that window as signal. The remaining points are now considered to be noise.

The baseline signal is obtained by either using linear interpolation (default) or fitting a cubic smoothing spline taking only the noise. The baseline can be shifted up or down by using the parameter `threshold`, which is done by the formula:

$$B' = B + 4\sigma(t - 0.5)$$

where B is the fitted spline, σ the standard deviation of the noise, and t is the `threshold` between 0 and 1. Finally, the corrected signal is calculated by subtracting B' to the original signal.

Value

The output depends on whether the input peaks is a matrix or a list. If it is a matrix, then the function returns a matrix of the same dimensions with the baseline corrected intensities. If instead peaks is a list, then the element called "Peaks" will hold the output.

Note

This function is intended to be run internally, but it is exported for advanced users.

Author(s)

Alvaro Cuadros-Inostroza

References

David Chang, Cory D. Banack and Sirish L. Shah, Robust baseline correction algorithm for signal dense NMR spectra. *Journal of Magnetic Resonance 187 (2007) 288-292*

See Also

RIcorrect, baseline

Examples

```
# get a random sample CDF from TargetSearchData
require(TargetSearchData)
cdffile <- sample(tsd_cdffiles(), 1)
pdata <- peakCDFextraction(cdffile)

# restrict mass range to reduce computing time (not needed for
# actual data)
pdata$Peaks <- pdata$Peaks[, 1:10] ; pdata$massRange <- c(85, 94)

# make a fake baseline as constant + noise (the CDF files have been
# already baseline corrected by the vendor software).
nscans <- length(pdata$Time)
noise <- as.integer(1000 + rnorm(nscans, sd=5))
pdata$Peaks <- pdata$Peaks + noise

# change parameters and see how the results change
pdata1 <- baselineCorrection(pdata)
pdata2 <- baselineCorrection(pdata, threshold = 1, alpha = 0.97)

# pick random trace k
k <- 6
m <- cbind(pdata$Peaks[, k] - noise, pdata1$Peaks[, k], pdata2$Peaks[, k])
matplot(pdata$Time, m, type='l', lty=1, xlab='time', ylab='intensity')
legend('topleft', c('original', 'base correct 1', 'base correct 2'),
       col=1:3, lty=1, lwd=1)
```

`baselineCorrectionQuant` *Baseline correction - quantiles method*

Description

This function perform baseline correction using a quantiles around a moving window algorithm.

Usage

```
baselineCorrectionQuant(peaks, time, smooth=0, qntl=0.50, width=30,
                        unit=c("seconds", "points"), steps=10)
```

Arguments

`peaks`	Either a matrix object of spectra peak intensities to be baseline corrected, where the rows are retention times and columns are mass traces; or, a named list containing an element called `"Peaks"` which such matrix and another called `"Time"` with the retention time in seconds. The list can be generated by `peakCDFextraction`
`time`	A vector of retention time in seconds. This parameter is used if `peaks` is a matrix. Otherwise, the element called `"Time"` is used instead and this parameter is ignored.
`smooth`	An integer. Smooth each signal by this number of points using a moving average. Smoothing is disabled if this value is less or equal than 1. Note that the smoothing is applied after the baseline correction.
`qntl`	Numeric scalar. The quantile for baseline estimation. The value must be in `[0, 1]`.
`width`	Numeric scalar. The size of the window centered around a scan for baseline estimation. The size depends on the parameter `unit` below.
`unit`	A string which chooses if the `width` are points (scans) or seconds.
`steps`	Integer scalar greater than zero. To speed up computation, the baseline algorithm does not compute the baseline estimate in each single scan, but in intervals of `steps` steps. The intermediate points are estimated by simple linear regression.

Details

Applies a quantile based baseline estimation method. The method is applied for each ion mass trace (column of peaks) individually. It simple computes for each data point of the trace the `qntl` quantile, for example the 50% quantile, i.e., the median, of all the points which are within a `width` distance or it.

In order for the method to work, select a `width` much larger than the widest peak.

For speed efficiency, and assuming that the baseline is a smooth curve, the quantiles are computed every `step` points. For example, if `step=3`, then the quantiles will be computed every third scan instead of every point. If instead `step=1`, then it will computed in every scan. The baseline of the points in between (if `step > 1`) are approximated by linear interpolation.

Value

Returns a list with the same elements as the input, but the element `"Peaks"` containing baseline corrected values. In case peaks is a matrix, it returns a matrix of the same dimension instead.

Author(s)

Alvaro Cuadros-Inostroza

See Also

RIcorrect, baseline, baselineCorrection

Examples

```
# get a random sample CDF from TargetSearchData
require(TargetSearchData)
cdffile <- sample(tsd_cdffiles(), 1)
pdata <- peakCDFextraction(cdffile)

# restrict mass range to reduce computing time (not needed for
# actual data)
pdata$Peaks <- pdata$Peaks[, 1:10] ; pdata$massRange <- c(85, 94)

# make a fake baseline as constant + noise (the CDF files have been
# already baseline corrected by the vendor software).
nscans <- length(pdata$Time)
noise <- as.integer(1000 + rnorm(nscans, sd=5))
pdata$Peaks <- pdata$Peaks + noise

# change parameters and see how the results change. Note that the default
# width of 30 seconds might be too small
pdata1 <- baselineCorrectionQuant(pdata, steps=5)
pdata2 <- baselineCorrectionQuant(pdata, width=50, steps=5)

# pick random trace k and compare correction values
k <- 6
m <- cbind(pdata$Peaks[, k] - noise, pdata1$Peaks[, k], pdata2$Peaks[, k])
matplot(pdata$Time, m, type='l', lty=1, xlab='time', ylab='intensity')
legend('topleft', c('original', 'base correct 1', 'base correct 2'),
       col=1:3, lty=1, lwd=1)
```

---

checkRimLim *Visually check retention index marker limits*

Description

A function to visually check if the retention time search limits of the retention index markers (aka FAMEs) are correct or need some adjustment.

Usage

```
checkRimLim(samples, rim, layout, show = TRUE, single = TRUE, extend = 0.5,
  rect.col = "#e7e7e7", mar = c(2, 2, 2, 2), oma = c(3, 3, 2, 0.5),
  cex.main = 1, type = "l", ...)
```

Arguments

`samples`	A `tsSample` object created by `ImportSamples`.
`rim`	A `tsRim` object describing the retention index markers. See `ImportFameSettings`.
`layout`	A vector of the form `c(nr, nc)` to arrange the panel by `nr` rows and `nc` columns. If missing then the layout is created automatically.
`show`	Logical. If `FALSE` the plot is not shown, but the data points can be used for further inspection or for custom plots.
`single`	Logical. If `TRUE` a *single* sample will be selected randomly for plotting. This was the old default behavior. If `FALSE` all samples will be used for plotting (but see note below).
`extend`	a numeric coefficient to extend the time window search of the respective time marker. Defaults to `0.5`.
`rect.col`	the color for the background rectangle which indicates the current retention time limits.
`mar`	the subplots margins, passed to `par()`.
`oma`	the outer plot margins, passed to `par()`.
`cex.main`	The magnification to be used for main titles, passed to `par()`.
`type`	A character vector indicating the type of plots. Default `"l"` for lines. Passed to `matplot()`.
`...`	extra plotting arguments passed to `matplot()` such as `col`, `lty`, `pch`, `lwd`.

Details

The function takes a `tsSample` object and creates a panel plot of the m/z traces around the area in which a marker is expected to be, one panel for each marker. By default, a single (random) sample is chosen (see option `single`) for plotting, but it is also possible to visualize many samples at the same time (but see note below).

For multiple sample visualization, it is recommended to use sub-setting for the `samples` and `rim` arguments in order to avoid over crowded plots, specially when there are several samples and several retention index markers. See the examples below.

Plotting options such as `col`, `lty`, `pch`, `lwd` can be passed; these are in turn passed to `matplot()`. Note that these are passed to each panel; it not possible to pass different options to different panels (for example to have different line colors). Moreover, using options such as `xlim` and `ylim` may result in ‘empty’ plots (because each panel needs its own limits). If different styles are required, then either make your own function from the output or use subsets of the `rim` (`tsRim`) object.

Value

The output value is either `invisible` or it depends on the option `single`. If this option is `TRUE` (the old behavior), then the output will be a list of `n` times `2` matrices, each element corresponding to a retention marker. Columns are retention time and intensities of the respective marker's *m/z*. The rows can be as many data points are within the search window.

If `single=FALSE`, the output is a list whose length is equal to `length(samples)` and its names are equal to `sampleNames(samples)`. Each element is in turn a list with exactly the same structure described in the paragraph above.

Note

If `single=TRUE`, all `CDF` files will be scanned, which can take a significant amount of time to parse, in particular, if there are hundred of them. Also due to this, each panel of the plot could be extremely crowded. It is therefore recommended to use sample sub-setting to reduce this number, as shown in the examples below.

See Also

`matplot` for plotting parameters, `par` for inner and outer margins parameters, `tsSample`, `sampleNames`, `tsRim`, `ImportFameSettings`

Examples

```
require(TargetSearchData)

# get the cdf path TargetSearchData
cdfpath <- tsd_data_path()

# import samples (see ImportSamples() for details)
samples <- ImportSamples(tsd_file_path("samples.txt"), CDFpath = cdfpath)

# Import RI markers (see ImportFameSettings())
rim <- ImportFameSettings(tsd_file_path("rimLimits.txt"))

# choose a sample at random and plot the m/z traces around the retention time window
ret <- checkRimLim(samples, rim)

# to choose a specific sample and marker, use subsetting
ret <- checkRimLim(samples[3], rim[1:2])

# to display many samples at the same time, set the option `single` to `FALSE` and select a subset
# of samples (recommended). In this example the first three samples are chosen.
ret <- checkRimLim(samples[1:3], rim, single=FALSE)

# in general, visualizing all samples (as shown below) at the same time it is not recommended
# for large number of samples.
## Not run:
  ret <- checkRimLim(samples, rim, single=FALSE)

## End(Not run)
```

`FAMEoutliers` *FAME outlier detection*

Description

A function to detect retention time marker (FAME) outliers.

Usage

```
FAMEoutliers(samples, RImatrix, pdffile = NA, startDay = NULL, endDay = NULL,
             threshold = 3, group.threshold = 0.05)
```

Arguments

`samples`	A `tsSample` object created by `ImportSamples` function.
`RImatrix`	A retention time matrix of the found retention time markers.
`pdffile`	A character string naming a PDF file where the FAMEs report will be saved.
`startDay`	A character vector with the starting days of your day groups.
`endDay`	A character vector with the ending days of your day groups.
`threshold`	A standard deviations cutoff to detect outliers.
`group.threshold`	A numeric cutoff to detect day groups based on hierarchical clustering. Must be between `0..1`.

Details

If no `pdffile` argument is given, the report will be saved on a file called `"TargetSearch-YYYY-MM-DD.FAME-report.pd`
where `YYYY-MM-DD` is a date.

If both `startDay` and `endDay` are not given (both set to either `NULL` or `NA`), the function will try to detect day groups using a hierarchical clustering approach by cutting the tree using `group.threshold` as cutoff height. Otherwise, both must have the same length, must not contain `NA`s, and must match the measurement days of the object `samples`. See example below.

Retention time markers that deviate more than `threshold` standard deviations from the mean of their day group will be identified as outliers.

Value

A logical matrix of the same size of `RImatrix`. A `TRUE` value indicates that the retention time marker in that particular sample is an outlier.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`RIcorrect`, `ImportSamples`, `TSExample`

Examples

```
# load pre-calculated example data and objects
data(TSExample)

# find the retention marker outliers of the example data and save it in "outlier.pdf"
outliers <- FAMEoutliers(sampleDescription, RImatrix, pdffile = "outlier.pdf")

# find the outliers (although they are reported in the output PDF file)
apply(outliers, 1, which)

# using `startDay` and `endDay` to set day groups manually
sampleDays(sampleDescription) <- rep(letters[1:3], each=5) # make fake days a, b, c

# set days 'a' and 'b' in a group and day 'c' by itself
startDay <- c('a', 'c')
endDay <- c('b', 'c')
```

```
outliers <- FAMEoutliers(sampleDescription, RImatrix, pdffile="outlier.pdf",
             startDay=startDay, endDay=endDay)
```

FindAllPeaks   *Extract peaks from chromatogram files - low level function*

Description

This function extracts all peaks of a given metabolite in a given retention time (RT) or retention index (RI) window. This function is intended for fine-tuning metabolite search parameters.

Usage

```
FindAllPeaks(samples, Lib, libID, dev=NULL, mz=NULL, RI=NULL, RT=NULL,
          mz_type = c('selMass', 'quantMass', 'topMass'),
          columns = NULL)
```

Arguments

samples	A tsSample object created by ImportSamples function.
Lib	A tsLib object created by ImportLibrary function.
libID	An index (integer or character) value representing the respective metabolite in the reference library Lib. It must be a scalar.
dev	The allowed retention index (RI) deviation or NULL. See details below.
mz	A list of m/z values to search or NULL.
RI	The expected retention index or NULL.
RT	The expected retention time to search for instead of retention index, or NULL.
mz_type	whether to search for the selective, quantification or top masses of the respective metabolite.
columns	Either NULL, a character vector, or an integer vector. In most cases, leave it as NULL. This parameter is used to configure the column names or positions of RI text files. See the documentation on the text2bin function for further details.

Details

The function searches for all peaks of a metabolite in all samples within a time window (RT or RI). The parameters dev, mz, RI, and RT have preference over the settings of the metabolite indexed by libID. Moreover, the parameter RI has preference of over RT (see below). In fact, if all of these parameters are not NULL, then refLib and libID are not used.

The metabolite search can be performed using retention index (RI) or retention time (RT). To search using RI, either specify the value by setting RI, or set both RI and RT to NULL, which defaults to searching by the library RI. To search using RT, you **must** set RI to NULL and set dev to a value (i.e., not NULL), otherwise an error will be thrown. See examples below.

The dev parameter can either be NULL (as stated above), or a numeric vector. If the length is equal to 1, then the RI search window is the given RI plus or minus dev. If the length is 2, then the search window is from RI + dev[1] to RI + dev[2]. **Note** than in this case dev[1] is usually a negative value. An error is raised if the length is greater than 2. Note that as mentioned above, this parameter is required to search by RT. Needless to say, the `dev` parameter units should correspond with the units of `RI` or `RT` accordingly.

The `columns` parameter is only needed for custom text RI files. There is (usually) no need to change it.

Value

It returns a matrix in which each row represent a hit. Note that there can be zero rows if no hits are found. The columns are named and these are:

`Int`	Peak intensity
`RI`	Retention Index
`RI`	Retention Time
`mz`	the searched m/z value
`fid`	the respective file or sample index. An integer value.

Note

This is an internal function not intended to be invoked directly, but it is exposed for convenience and advanced users.

In the future it may replace `FindPeaks`.

See also the function `ri_data_extract` which offers a similar functionality but with different input parameters.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`FindPeaks`, `ri_data_extract`,

Examples

```
# load pre-calculated example data files and objects
require(TargetSearchData)
data(TSExample)

# get and set the RI file path
RIpath(sampleDescription) <- tsd_data_path()

# search all peaks of Valine (GC.3) and selective masses
peaks <- FindAllPeaks(sampleDescription, refLibrary, 'GC.3')
head(peaks)

# a numeric index is also allowed
peaks <- FindAllPeaks(sampleDescription, refLibrary, 3)
head(peaks)

# an asymmetric deviation search
peaks <- FindAllPeaks(sampleDescription, refLibrary, 'GC.3', dev=c(-1000, 2000))
head(peaks)
```

```
# search arbitrary masses at arbitrary RI. The reference library and ID
# must be set set to NULL.
peaks <- FindAllPeaks(sampleDescription, NULL, NULL, dev=3000, RI=270000, mz=c(144, 100))
head(peaks)

# to search for RT, set the RT parameter to not NULL
peaks <- FindAllPeaks(sampleDescription, refLibrary, 'GC.3', dev=3.0, RT=250)
peaks <- FindAllPeaks(sampleDescription, NULL, NULL, dev=3.0, RT=250, mz=c(144, 100))

## Not run:
    # note that 'dev' is required; this throws an error
    peaks <- FindAllPeaks(sampleDescription, NULL, NULL, RT=250, mz=c(144, 100))

## End(Not run)
```

`FindPeaks` *Extract peaks from chromatogram files*

Description

This function extracts the maximum intensity of a list of masses in a given RI window.

Usage

```
FindPeaks(my.files, refLib, columns = NULL, showProgressBar = FALSE)
```

Arguments

`my.files`	A character vector naming RI files to be searched.
`refLib`	A numeric matrix with three columns or a list of three column matrices. The second column contains the masses and the first and third column contains the RI limits.
`showProgressBar`	Logical. Should the progress bar be displayed?
`columns`	Either `NULL`, a `character` vector, or an `integer` vector. In most cases, leave it as `NULL`. This parameter is used to configure the column names or positions of RI text files. See the documentation on the `text2bin` function for further details.

Details

The reference library parameter `refLib` can be either a single three-column matrix or a list of such matrices. If it is a list, the length must match the length of `my.files`. In this case, every component will be used to iteratively search in the corresponding file.

The RI files format can be either "text" or "binary". The type is detected dynamically.

Value

A `tsMSdata` object.

Note

This is an internal function not intended to be invoked directly.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`medianRILib`, `sampleRI`, `peakFind`, `tsMSdata`

Examples

```
# load example CDF files
require(TargetSearchData)
# load pre-calculated example data and objects
data(TSExample)

# get RI file path
RI.path <- tsd_data_path()
# update RI file path
RIpath(sampleDescription) <- RI.path

my.files <- RIfiles(sampleDescription)
# make a three column matrix: lower RI, mass, upper RI
refLib   <- refLib(refLibrary)
head(refLib)

# extract the peaks
peaks <- FindPeaks(my.files, refLib)
```

`fixRI` *Fixing Retention Time Index Correction*

Description

This function can be used to correct the detected retention time index (RI) markers or to manually force their location to specific retention times if, for example, the RI markers were not co-injected with the biological samples.

Usage

```
fixRI(samples, rimLimits, RImatrix=NULL, sampleNames=NULL, quiet=TRUE)
```

Arguments

`samples`	A `tsSample` object created by `ImportSamples` function.
`rimLimits`	A `tsRim` object. See `ImportFameSettings`.
`RImatrix`	Optional. A retention time matrix of the found retention time markers that was obtained after running `RIcorrect`.
`sampleNames`	Optional. A character vector naming the samples that are to be RI corrected.
`quiet`	Logical. Do not print a list of converted files.

Details

Sometimes the retention index limits (see ImportFameSettings) are not set correctly and you will have to run the peak detection and RI correction function (RIcorrect) again, which may take a long time specially if there are many samples.

Instead, a simple approach is to fix the RI limits and use this function to correct the generated RI files. Since these files are much smaller than CDF files (chromatograms), this runs much faster.

Other possibility is that the time positions of one or more RI markers were wrongly detected because there was just simply no peak or the RI markers where not co-injected in some samples. You could manually force the locations of the RI markers. This case is discussed in the "RICorrection" vignette.

The behavior of this function depends on whether the parameter `RImatrix` is `NULL` or not. If `NULL`, the RI markers will be searched again (using the settings of `rimLimits` parameters, which you should have already fixed) and the resulting values will be used to correct the RI files. If it is a numeric matrix, then these values will be used to correct the RI files. Note that this matrix dimensions are exactly *m* samples (rows) times *n* (columns) RI markers.

`sampleNames` controls which samples will be corrected. If `NULL` then all samples will be corrected. It could be character vector (sample names) or a numeric vector representing the sample indexes.

Value

Invisible `NULL`. It prints the corrected samples if `quiet` is `FALSE`.

Note

In case the RI files are in text format and their column names are not standard (for example, when the files were generated with another software), use the global option `'TS_RI_columns'` or transform the RI files to TargetSearch binary format. See the documentation in function text2bin.

Author(s)

Alvaro Cuadros-Inostroza

See Also

RIcorrect, FAMEoutliers,ImportSamples, ImportFameSettings

Examples

```
require(TargetSearchData)
# import refLibrary, rimLimits and sampleDescription.
data(TSExample)
# get the CDF files
cdfpath <- tsd_data_path()

# select a subset of samples
smp <- sampleDescription[1:4]

# update the CDF path
CDFpath(smp) <- cdfpath

# make a copy of the RI markers object
rim <- rimLimits
```

```
# mess up the limits of marker 3 (real value is 369 seconds app.)
rimLimits(rim)[3,] <- c(375, 400)

# run RIcorrect (skip CDF-4 conversion)
RImat <- RIcorrect(smp, rim, writeCDF4path=FALSE,
           Window = 15, IntThreshold = 50)

# fix the limits of marker 3
rimLimits(rim)[3,] <- c(360, 400)

# you could run again RIcorrect, but this is faster
fixRI(smp, rim)

# get the RI matrix
RImat <- riMatrix(smp, rim)

# compare the values with the real ones (previously stored in RImatrix)
stopifnot( all.equal(RImat[, 1:4], RImatrix[,1:4], tolerance=1e-8) )

# manual adjustment or RI markers for sample 3.
# Warning: this is just an example to illustrate how to use this function.
#          don't do this unless you know what you're doing.
RImat[,3] <- c(252, 311, 369)
fixRI(smp, rim, RImat, 3)
```

`ImportFameSettings` *Retention time markers settings*

Description

This function imports a list of retention standard markers for RI correction.

Usage

```
ImportFameSettings(file, mass=NULL, standard=NULL, colnames=FALSE, ...)
```

Arguments

`file`	A character string naming a tab-delimited text file with standard markers OR a matrix-like object.
`mass`	Optional. The m/z standard marker(s). Either a scalar, which applies to every marker, or a vector.
`standard`	Optional. The RI values of the standards. A numeric vector.
`colnames`	Logical flag. If `TRUE`, the column names will be used to look up for the settings, otherwise the column positions will be used. See details below.
`...`	Options passed to `read.delim`, which is used internally to read the settings file.

Details

The standard marker file is a tab-delimited text file with 2, 3 or 4 columns. Additional columns are ignored. If the parameter `colnames` is `FALSE`, then the column names are ignored and they must be in the following specified below. If `TRUE`, use the column names below (case is ignored). Note that some columns are optional as they can be specified by the options `mass` and `standard`.

- LowerLimit - The Retention time lower limit in seconds (required).
- UpperLimit - The Retention time upper limit in seconds (required).
- RIstandard - The RI value of that standard. This is optional (option standard).
- mass - The m/z standard marker. This is optional (option mass).

Instead of passing a file via the file option, one can pass a matrix or a data.frame, The same aforementioned conditions apply in terms of column names and number.

In general, the mass and the standard should be to NULL. If not, then they take precedence over the values defined by the file parameter.

If no arguments are given, a default object will be returned.

Value

A tsRim object.

Note

If the parameter colnames is set to TRUE, then care needs to be taken regarding the column names: they must be named exactly as described in the *details* section, otherwise they can be silently ignored and a default value might be used instead. This has been the default behavior.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

RIcorrect, tsRim, read.delim

Examples

```
require(TargetSearchData)
# get the RI marker definition file
rim.file <- tsd_file_path("rimLimits.txt")

# set the mass marker to 87
mass <- 87

# load the definition
rimLimits <- ImportFameSettings(rim.file, mass = mass)

# sometimes you need to change the limits of a particular standard
rimLimits(rimLimits)[2,] <- c(410, 450)

# to change the mass value
rimMass(rimLimits) <- 85

# alternatively, you can pass a two column matrix as limits and pass the
# rest as parameters separately (toy example).
limits <- cbind(c(10, 20, 30), c(15, 25, 35))
mass <- 100
standard  <- c(100, 200, 300)
rimLimits <- ImportFameSettings(limits, mass, standard)
```

```
# or combine all into a matrix (note the column order!)
def <- cbind(limits, standard, mass)
rimLimits <- ImportFameSettings(def)
```

`ImportLibrary` *Library import*

Description

These functions import a metabolite library file that will be used to processed the GC-MS data. Two file formats are supported: a tab-delimited format and the more common NIST MSP format.

Usage

```
ImportLibrary(x, type = c("auto", "tab", "msp"), ...)

ImportLibrary.tab(libfile, fields = NULL, RI_dev = c(2000,1000,200),
    SelMasses = 5, TopMasses = 15, ExcludeMasses = NULL,
    libdata, file.opt=NULL)

ImportLibrary.msp(libfile, fields = NULL, RI_dev = c(2000,1000,200),
    SelMasses = 5, TopMasses = 15, ExcludeMasses = NULL)
```

Arguments

`x`	A character string or a `data.frame`. If `data.frame`, it will be passed to `ImportLibrary.tab` as parameter `libdata`. If `character`, it will be passed as `libfile` to either `ImportLibrary.tab` or `ImportLibrary.msp` according to the file type (option `type`).
`libfile`	A character string naming a library file. See details.
`type`	The library file format. Possible options are `"tab"` for a tab-delimited file, `"msp"` for NIST MSP format, or `"auto"` for autodetection. Default to `"auto"`.
`fields`	A two component list. Each component contains a regular expression used to parse and extract the fields for retention index and selection masses. Only meaningful for MSP format.
`RI_dev`	A three component vector with RI windows.
`SelMasses`	The number of selective masses that will be used.
`TopMasses`	The number of most intensive masses that will be taken from the spectrum, if no `TOP_MASS` is provided.
`ExcludeMasses`	Optional. A vector containing a list of masses that will be excluded.
`libdata`	Optional. A data frame with library data. The format is the same as the library file. It is equivalent to importing the library file first with `read.table` and calling `ImportLibrary.tab` after. This might be preferable for "fine tuning", for example, if the library file is in CSV format instead of tab-delimited.
`file.opt`	Optional. A list containing arguments to be passed to `read.table`.
`...`	Further arguments passed to `ImportLibrary.tab` or `ImportLibrary.msp`

Details

`ImportLibrary` is a wrapper for functions `ImportLibrary.tab` and `ImportLibrary.msp` which detects automatically which function should be called.

`ImportLibrary.tab` reads a tab delimited text file by calling the function `read.table` which will be parsed and converted to a `tsLib` object. The following arguments are used by default (which are not exactly the defaults for `read.table`):

`header=TRUE, sep="\t", quote="", dec=".", fill=TRUE, comment.char="#"`

The argument `file.opt` can be used to change these options. Other alternative is to import first the file with `read.table` and friends, and call `ImportLibrary` with the resulting `data.frame`. This allows more flexibility with libraries with unusual characters, for example.

These columns are needed:

- `libID` - A unique identifier for the metabolite.
- `Name` - The metabolite name.
- `RI` - The expected RI.
- `SEL_MASS` - A list of selective masses separated with semicolon.
- `TOP_MASS` - A list of the most abundant masses to be searched, separated with semicolons.
- `Win_k` - The RI windows, k = 1,2,3. Mass search is performed in three steps. A RI window required for each one of them.
- `SPECTRUM` - The metabolite spectrum. *m/z* and intensity values are separated by spaces, colons or semicolons (see notes on the format below).
- `QUANT_MASS` - A list of masses that might be used for quantification. One value per metabolite and it must be one of the selective masses. (optional)

The columns `Name` and `RI` are mandatory. At least one of columns `SEL_MASS`, `TOP_MASS` and `SPECTRUM` must be given as well. By using the parameters `SelMasses` or `TopMasses` it is possible to set the selective masses or the top masses from the spectra. The parameter `ExcludeMasses` is used only when masses are obtained from the spectra. The parameter `RI_dev` can be used to set the RI windows. Note that in this case, all metabolites would have the same RI windows.

The `SPECTRUM` format can be a list of *m/z*-intensity pairs separated by colons: `85:8 86:13 87:5 88:4 ...`, or similar to the MSP spectrum format (shown below): `85 8; 86 13; 87 5; 88 4; ...`. The internal parser is **not** strict, so even a list of integers separated by spaces is allowed, for example, `86 8 86 13 ...`. The only restriction is that the number of integers is even, otherwise the input is truncated. If the spectrum is not available for some analytes, simply leave the respective cell empty or use `NA`. **Note:** Only digits(0-9), spaces, colons(:) and semicolons(;) are allowed in this column; the presence of other characters will generate and error, unless it is `NA`.

A unique identifier may be provided as well (`libID`). This could be a external database identifier. If it is not provided, a random identifier will be created of the form `GC.k`, k = 1,...,N.

The MSP format is a text file that can be imported/exported from NIST. A typical MSP file looks like this:

```
Name: Pyruvic Acid
Synon: Propanoic acid, 2-(methoxyimino)-, trimethylsilyl ester
Synon: RI: 223090
Synon: SEL MASS: 89|115|158|174|189
Formula: C7H15NO3Si
MW: 189
Num Peaks: 41
```

```
  85    8;  86   13;  87    5;  88    4;  89  649;
  90   55;  91   28;  92    1;  98   13;  99  257;
 100  169; 101   30; 102    7; 103   13; 104    1;
 113    3; 114   35; 115  358; 116   44; 117   73;
 118   10; 119    4; 128    2; 129    1; 130   10;
 131    3; 142    1; 143   19; 144    4; 145    1;
 157    1; 158   69; 159   22; 160    4; 173    1;
 174  999; 175  115; 176   40; 177    2; 189   16;
 190    2;

Name: another metabolite
...
```

Different entries must be separated by empty lines. In order to parse the retention time index (RI) and selective masses (SEL MASS), a two component list containing the field names of RI and SEL_MASS must be provided by using the parameter `fields`. In this example, use `field = list("RI: ", "SEL MASS: ")`. Note that `ImportLibrary` expects to find those fields next to "Synon:". Alternatively, you could provide the RI and SEL_MASS using the `tsLib` methods.

Libraries for TargetSearch and for different retention index systems, such as VAR5 or MDN35, can be downloaded from `http://gmd.mpimp-golm.mpg.de/`.

Value

A `tsLib` object.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ImportSamples`, `tsLib`

Examples

```
# get the reference library file
require(TargetSearchData)
lib.file  <- tsd_file_path("library.txt")

# Import the reference library
refLibrary <- ImportLibrary(lib.file)

# set new names for the first 3 metabolites
libName(refLibrary)[1:3] <- c("Metab01", "Metab02", "Metab03")

# change the retention time deviations of Metabolite 3
RIdev(refLibrary)[3,] <- c(3000,1500,150)
```

`ImportSamples` *Sample definitions*

Description

These functions import sample information from either a tab delimited file or a file system path. The sample information contain sample names, CDF files and their measurement day, also known as, measurement run or measurement batch.

Usage

```
ImportSamples(sampfile, CDFpath, RIpath, ftype=c("binary", "text"), ...)

ImportSamplesFromDir(CDFpath=".", RIfiles=FALSE, ftype=c("binary", "text"),
  verbose=FALSE, ...)
```

Arguments

`sampfile`	A character string naming a sample file or a `data.frame`. See details.
`CDFpath`	A character string naming a directory where the CDF files are located (by default is the current directory)
`RIpath`	A character string naming a directory where the RI corrected text files are/will be located.
`RIfiles`	Logical. If `TRUE`, the function will look for for RI files (RI_*) instead of CDF files (the default).
`ftype`	A character string giving the file type for RI files. Options are "binary" and "text".
`verbose`	Logical. Show more output if `TRUE`.
`...`	Extra options whose meaning depend on the function: for `ImportSamples`, the options will be passed to `read.delim`, while for `ImportSamplesFromDir`, these are passed to `dir`

.

Details

The sample file is a tab-delimited text file or, alternatively, a `data.frame` like the one would get by calling `read.delim`. The following columns are expected, though only the first one below is mandatory.

- `CDF_FILE` - The list of baseline corrected CDF files (mandatory).
- `MEASUREMENT_DAY` - The day or batch when the sample was measured (optional).
- `SAMPLE_NAME` - A unique sample identifier (optional).

The function first looks for columns matching those names. If they are not found, then it looks for columns with the substrings 'cdf', 'day' and 'name' respectively. If the 'cdf' column is not found, the function raises an error, but if 'day' is not found, then it assumes the measurement day is the same for all. Moreover, if the column for sample names/identifiers is missing, then the function uses the 'cdf' file names as identifiers.

During the column matching, the first match is taken in case of ambiguity. The column order does not matter. Other columns could be included in that file. They won't be used by the script, but will be included in the "sample" R object.

The files given in the 'cdf' column may optionally include a relative path, such as `mypath/myfile.cdf`, which will be relative to the working directory or relative to the argument `CDFpath`. Optionally, the file extension can be omitted because 'TargetSearch' adds the `.cdf` extension automatically if missing. Note: it may fail in case sensitive file systems as a lower case extension is appended.

If the parameter `CDFpath` is missing, then the current directory is used. If `RIpath` is missing, the value of `CDFpath` will be used.

The `ftype` parameter sets the file format of the RI files, which are created by the function `RIcorrect`. "text" is the *classic* text format and "binary" is a binary version, designed for speed, of the text format (TargetSearch >= 1.12.0). The file format can be identified by the file extension (".txt" for text, ".dat" for binary), but this is just an indicator: the file format is detected dynamically during file reading. Use the method `fileFormat` to set or get this parameter.

The function `ImportSamplesFromDir` scans for cdf files in the current or given directory. The search can be made recursive by passing the parameter `recursive=TRUE` (actually passed to `dir`).

The parameter `verbose` will print a list of detected files is set to `TRUE`. This can be used for debugging.

Value

A `tsSample` object.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ImportLibrary`, `tsSample`

Examples

```
# get the sample definition definition file
require(TargetSearchData)
cdfpath <- tsd_data_path()
sample.file  <- tsd_file_path("samples.txt")

# set a path where the RI files will be created
RIpath <- "."

# import samples
sampleDescription <- ImportSamples(sample.file, CDFpath = cdfpath, RIpath = RIpath)

# change the sample names. It is required that the names must be unique.
sampleNames(sampleDescription) <- paste("Sample", 1:length(sampleDescription), sep = "_")

# change the file paths (relative to the working path)
CDFpath(sampleDescription) <- "my_cdfs"
RIpath(sampleDescription)  <- "my_RIs"
```

`makeIndex` *Make an index vector out of selective/top masses*

Description

A function that makes an index for selective or top masses. Replaces the method `libId`.

Usage

```
makeIndex(lib, sel=TRUE)
```

Arguments

`lib`	A tsLib object.
`sel`	A logical flag. If `TRUE` return an integer index for selective masses, otherwise for top masses.

Details

This is an internal function that should not be used. It is exposed because the method it replaces, `libId`, was originally exposed, so compatibility is not lost.

Value

An integer vector, where each number is associated with an element of the library object, and its multiplicity with the number of selective or top masses.

Note

It is equivalent to the deprecated method `libId`.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`tsLib`

Examples

```
# load example objects
data(TSExample)
# generate an index for selective masses
(idx <- makeIndex(refLibrary, TRUE))
```

`medianRILib` *Median RI library correction*

Description

Return a `tsLib` object with the median RI of the selective masses across samples.

Usage

```
medianRILib(samples, Lib, makeReport = FALSE, pdfFile = "medianLibRep.pdf",
             columns = NULL, showProgressBar = FALSE)
```

Arguments

`samples`	A `tsSample` object created by `ImportSamples` function.
`Lib`	A `tsLib` object created by `ImportLibrary` function.
`makeReport`	Logical. If `TRUE` will report the RI deviations for every metabolite in the library.
`pdfFile`	The file name where the report will be saved.
`columns`	Either `NULL`, a `character` vector, or an `integer` vector. In most cases, leave it as `NULL`. This parameter is used to configure the column names or positions of RI text files. See the documentation on the `text2bin` function for further details.
`showProgressBar`	Logical. Should the progress bar be displayed?

Value

A `tsLib` object. It will update the slot `med_RI` which contains the median RI of every searched metabolite.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ImportSamples`, `ImportLibrary`, `tsLib-class`

Examples

```
require(TargetSearchData)
data(TSExample)

# get RI file path
RI.path <- tsd_data_path()
# update RI file path
RIpath(sampleDescription) <- RI.path
# Import Library
refLibrary        <- ImportLibrary(tsd_file_path('library.txt'))
# update median RI
refLibrary        <- medianRILib(sampleDescription, refLibrary)
```

```
# perhaps you need to adjust the library RI of one metabolite and the allowed time
# deviation (first time deviation window)
libRI(refLibrary)[5] <- 306500
RIdev(refLibrary)[5,1] <- 2000

refLibrary       <- medianRILib(sampleDescription, refLibrary)
```

---

ncdf4Convert,tsSample-method
Method for converting CDF-3 files to CDF-4

Description

ncdf4Convert is a high level method used to convert from CDF-3 to 'TargetSearch' new CDF-4 custom file format.

Usage

```
ncdf4Convert(obj, path, ...)
```

Arguments

obj	A tsSample object
path	A character vector representing file path in which the newly created CDF-4 files will be stored. If missing, the files will be save in the same directory as the source CDF-3 files
...	Extra arguments passed to ncdf4_convert, such as baseline correction options.

Details

This is a high level interface to ncdf4_convert which uses a tsSample as input. The advantage of using this function is that it updates the paths to the CDF-4 files for downstream analysis.

The parameter path can be used to change the original paths, which may be necessary if those files are on a read-only file-system. If this parameter is missing, the same paths are used. Note that the path are re-cycled to match the length of the samples.

The ... can be used to pass extra arguments to ncdf4_convert and to baseline. Refer to those man pages for details.

Value

A new tsSample object with updated CDF-4 file paths. The files are converted in the background.

Author(s)

Alvaro Cuadros-Inostroza

See Also

ncdf4_convert

Examples

```
require(TargetSearchData)
data(TSExample)
# get the CDF files
cdfpath <- tsd_data_path()

# update the CDF path
CDFpath(sampleDescription) <- cdfpath

# transform the CDF (the files are copied in the current directoru)
newSamples <- ncdf4Convert(sampleDescription, path=".")
```

`ncdf4_convert` *Convert from a NetCDF file format 3 to format 4*

Description

Convert from NetCDF format 3 into a custom TargetSearch NetCDF format 4. The new NetCDF just contains the raw data in a matrix format in order to allow easier and faster data manipulation.

Usage

```
ncdf4_convert(cdfFile, outFile = NULL, force = FALSE,
  baseline = FALSE, ...)
```

Arguments

`cdfFile`	The NetCDF file to be converted
`outFile`	The new output file. If `NULL`, it replaces the `cdfFile`'s file extension (which should be `.cdf`) by `.nc4`. If the file extension is not `.cdf`, then `.nc4` is just appended. If the path to the file does not exist, it will be created automatically.
`force`	Logical. Set to `TRUE` to allow file overwrites, for example if the destination file still exists, in which case a warning is thrown. Default to `FALSE`.
`baseline`	Logical. Whether or not baseline correct the input file.
`...`	extra options passed to `baseline()`.

Details

Starting from version 1.42.0, TargetSearch introduces a custom NetCDF file which is used for faster and easier data manipulation. This means, ion traces within a retention time can be quickly extracted, which if often required before plotting. Formerly, this process required parsing the whole file before the data could be extracted.

Note that this function only takes one file at the time. To convert many files at the same time, see the function `ncdf4_convert_from_path()` or the high level method `ncdf4Convert()`. Alternatively, you can call this function in a loop or using the `lapply` family of functions.

Keep in mind this function is intended for internal use (or advanced users); it is exported for convenience. Using the method `ncdf4Convert()` is recommended.

Value

A string. The path to the converted file or invisible.

File structure

The structure of the NetCDF format 4 is straightforward and the variables and attributes are self-evident. The following variables are defined.

- `retention_time` is a vector representing the retention time in seconds (double).
- `retention_index` is a vector representing the retention time indices (double). If missing, then the variable contains zeros. Its length is equal to the length of `retention_time`.
- `mass_range` is vector of length two containing the minimum and maximum m/z values (integer).
- `intensity` is matrix of intensity values (integer) where columns represent ion traces and rows are scans. The dimensions are length of "retention time" times the number of ions, i.e., mass max - mass min + 1.

In addition, the following attributes are defined. Note that only `creator` and `version` are mandatory.

- `creator` a string equal to "TargetSearch" (for identification purposes).
- `version` file format version (string). Currently "1.1".
- `time_corrected` (optional) a flag (short integer) to indicate RI correction. If missing it defaults to `false`.
- `baseline_corrected` (optional) a flag (short integer) to indicate that the file was baseline corrected by TargetSearch. If missing it defaults to `false`.

Note

Currently, it is not possible to reconstruct the original NetCDF file from the converted file, especially if nominal mass or baseline correction was applied. On the other hand, if the NetCDF files are exported from custom chromatogram files (such as thermo raw files or LECO peg files), then the NetCDF 3 files can be deleted safely as there is always a way to recover them.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`ncdf4Convert()`, `ncdf4_convert_from_path()`, `baseline()`

Examples

```
require(TargetSearchData)

# get files from package TargetSearchData
cdfpath <- tsd_data_path()

# choose any file
cdf <- file.path(cdfpath, '7235eg04.cdf')
nc4 <- '7235eg04.nc4' # save file in current path

# run the function
ret <- ncdf4_convert(cdf, nc4)

# the output should match the output file
```

```
stopifnot(ret == nc4)

# Use mapply to convert many files at the same time.
cdf <- paste0('7235eg0', 6:8, '.cdf')
nc4 <- paste0('7235eg0', 6:8, '.nc4')
ret <- mapply(ncdf4_convert, file.path(cdfpath, cdf), nc4)
stopifnot(ret == nc4)
```

`ncdf4_convert_from_path`
Convert CDF files to CDF4 from a path automatically

Description

Scan and convert NetCDF format 3 files into a custom TargetSearch NetCDF format 4 automatically. The function scans the given input path(s) for files with `"cdf"` extension. This function simply wraps around `ncdf4_convert()`, so that function will then convert each detected file.

Usage

```
ncdf4_convert_from_path(cdf_path = ".", out_path = cdf_path, recursive = FALSE, ignore.case = TRUE,
```

Arguments

`cdf_path`	Character string. The path(s) to scan for CDF files.
`out_path`	Character string. The output path(s) in which the converted files will be saved.
`recursive`	Logical. Should the search recurse into directories?
`ignore.case`	Logical. Should the file extension matching be case insensitive?
`...`	Extra options passed to `ncdf4_convert()`, which in turn can be passed to `baseline()`.

Details

This function simply searches for CDF-3 files and converts them to CDF-4 (custom format). By default the current path is scanned and files are converted and save in the same place.

Multiple input directories can be given. If this is the case, the output path `out_path` must be either a single path (in which case all files will be copied to the same directory) or an equal number of paths, such that there is a corresponding match between input and output directories. If not, an assertion error will be raised.

By default the search is case insensitive (see parameter `ignore.case`). For sensitive case matching, the function looks for all lowercase extension `cdf`. At the moment, it is not possible to change this pattern.

If `recursive` is `TRUE`, the search will also descend into subdirectories and the converted files will follow a similar directory structure. Output directories will be created on demand if required (this is actually done by `ncdf4_convert()` internally).

Value

A character vector of generated files or invisible.

Author(s)

Alvaro Cuadros-Inostroza

See Also

ncdf4_convert(), baseline()

Examples

```
# in this example we'll use a temporary directory as source
# and storage of CDF (because we need to reduce testing time in Bioc)

# get some files from package TargetSearchData and copy them
# to `srcdir`
require(TargetSearchData)
cdffiles <- tsd_cdffiles()[1:4]
srcdir <- file.path(tempdir(), 'gc-ms-data')
dir.create(srcdir)
file.copy(cdffiles, srcdir)

# convert to output directory set above
outdir <- tempfile()
ncdf4_convert_from_path(srcdir, outdir)

# recursive search. note that output files will be saved under "gc-ms-data"
# in the output directory
cdfpath <- tempdir() # (base path where we copy the CDF files)
ncdf4_convert_from_path(cdfpath, outdir, recursive=TRUE)

# clean-up the mess
unlink(c(srcdir, outdir), recursive=TRUE)
```

---

ncdf4_data_extract   *Extract data ranges from a NetCDF file format 4*

Description

A flexible and convenient function to extract raw data from a NetCDF file format 4 using time ranges and m/z ranges or values. This is a better (and faster) alternative to the old peakCDFextraction() function, which reads the whole CDF into memory, specially when only sections of the CDF are needed.

Usage

```
ncdf4_data_extract(cdfFile, massValues, timeRange, useRT = FALSE)
```

Arguments

cdfFile	A path to a NetCDF file format 4.
massValues	A numeric vector representing m/z values.
timeRange	A numeric vector of length 2 representing the lower and upper time limits.
useRT	Logical. If TRUE, the time range is in seconds, otherwise if FALSE (default), the time range is in retention time index units (TRUE).

Details

The function takes a NetCDF format 4 generated by "TargetSearch" and extracts raw intensity values from given m/z ion traces within a given time range. The time range can be in seconds or arbitrary retention time index units. For the latter case, the function expects a time corrected CDF file.

If the given time range is out of range, a `NULL` value will be returned. In contrast, if the m/z values are out of range, then zeros will be returned for out of range masses (provided that the time range is not out of range). If `timeRange` is missing, then the whole time range. Similarly, if `massRange` is missing, then all the masses are extracted. If both are missing, the function behaves as `peakCDFextraction()`, but the output (a named list) uses slightly different names.

The NetCDF must be have been previously converted to the custom "TargetSearch" format, otherwise an error will be raised. See `ncdf4_convert()` for the conversion.

Value

A named list with the following components.

- `Time` Numeric vector: the retention time in seconds
- `Index` Numeric vector: the retention time index (or zero if the file was was not retention time corrected
- `Intensity` Matrix: Rows are the retention times (or scans) and columns are masses.
- `massRange` Numeric vector of length 2: the mass range of the CDF

Note

An error will be produced for invalid files. Also, this function is intended to be used internally, but it is exposed for convenience, for example, to create custom plots.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`ncdf4_convert()`, `peakCDFextraction()`

Examples

```
# If you already have a NCDF4 file set the variable accordingly
# and skip the following section
# nc4file <- "/path/to/netcdf4.nc4"

### create NCDF4 from NCDF3 from the package TargetSearchData
require(TargetSearchData)
# pick first file
cdffile <- tsd_cdffiles()[1]
nc4file <- tempfile(fileext=".nc4")
ncdf4_convert(cdffile, nc4file)

# update RI data using pre-computed values
ncdf4_update_ri(nc4file, c(252, 311, 269), c(262320, 323120, 381020))
### end
```

```
# extract all data (behaves like peakCDFextraction)
data <- ncdf4_data_extract(nc4file)

# extract only certain m/z values
data <- ncdf4_data_extract(nc4file, massValues=c(116, 192))

# to use mass ranges, use the colon (:) operator for example
data <- ncdf4_data_extract(nc4file, massValues=c(120:130, 200, 203:209))

# restrict extraction to a retention index interval
data <- ncdf4_data_extract(nc4file, massValues=c(116, 192),
                           timeRange=c(200000, 220000))

# same, but using retention time in seconds.
data <- ncdf4_data_extract(nc4file, massValues=c(116, 192),
                           timeRange=c(200, 300), useRT=TRUE)
```

---

ncdf4_plot_peak *Plot peaks from ncdf4 files*

Description

A simple function to plot *m/z* ion traces from a CDF-4 file. This new interface supersedes the function plotPeakSimple (but that can be used for CDF-3 files).

Usage

```
ncdf4_plot_peak(obj, massValues, timeRange, useRT = TRUE, showRT = useRT, plot = TRUE, ...)
```

Arguments

obj	Either a tsSample object or a path to NetCDF format 4 files.
massValues	A numeric vector representing *m/z* values.
timeRange	A numeric vector of length 2 representing the lower and upper time limits.
useRT	Logical. If TRUE (default), the time range is expressed in seconds, otherwise if FALSE, the time range is in retention time index units. This requires retention time corrected CDF files or an error will be raised.
showRT	Logical. Whether the *x*-axis of the resulting plot should represent seconds (TRUE) or retention index units (FALSE). This allows, for example, to search for a time range in seconds and display a graph in index units.
plot	Logical. If FALSE nothing is plotted. This option may be used to simply extract the data from the CDF files, but see also ncdf4_data_extract.
...	Extra arguments, such as pch, col, ..., to be passed to the plotting functions. See details below.

Details

The function takes a list of path to CDF-4 files or an object of class tsSample as input; a list a *m/z* values to select for; and a time range, which can be in seconds (default) or retention time index (RI) units. For the RI units to work, the CDF files must have been previously time corrected (See RIcorrect, updateRI).

The selection of the time range units depends on the parameter `useRT`. If `TRUE`, then the search will be restricted to the specified range in seconds, otherwise RI units will be used. However, the displayed time units are defined by the parameter `showRT`. The default behavior is to use the same time units that were search for, but using seconds to search and RI units to display is possible by setting `showRT = !useRT` (or the reverse). This is useful if the retention time correction was not performed correctly and you want to verify what retention time correspond to what retention index.

The logical parameter `plot` controls whether a plot is drawn. This might be useful for simply extracting the data used for the plot and not caring about the plot itself. The function `ncdf4_data_extract` can be used for that purpose as well (in fact that function is run under the hood).

Plot styling can be achieved by passing plotting extra parameters such as `col`, `pch`, `lty`, and others (see `par` and `plot`). There is however a caveat due to how this function is implemented: the function calls `matlines` once per each file (or sample), so parameters such as `col`, `lty`, etc., apply only to the *m/z* values, not to the samples.

If you want to display different color lines for different samples (for example), add a ‘s’ to the respective parameter name as a prefix. Currently, these parameters are allowed: `scol`, `stype`, `slty`, `slwd`, `spch`, `sbg`, `scex`. See the examples below for details.

Finally, all plotting parameters are cycled over and in case of conflicts, the ‘s’ parameters take precedence.

Value

Either a named list, one element per sample or CDF file, or `invisible`. Each element is in itself a list with the following components (same as the output of `ncdf4_data_extract`).

- `Time` Numeric vector: the retention time in seconds
- `Index` Numeric vector: the retention time index (or zero if the file was was not retention time corrected
- `Intensity` Matrix: Rows are the retention times (or scans) and columns are masses.
- `massRange` Numeric vector of length 2: the mass range of the CDF

Note

This function is only for CDF-4 files. Use the method `ncdf4Convert` or the function `ncdf4_convert` for conversion of CDF-3 files. You may also use function `plotPeakSimple` if you don’t want to convert the files.

Not all graphical parameters passed by `...` will work, in particular `panel.first` and `panel.last` are known to *not* work. Therefore, for more personalized plots, it is recommended to set `plot=FALSE` and use the returned data for plotting.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`plotPeak`, `plotPeakSimple`, `ncdf4_data_extract`, `RIcorrect`, `updateRI`

Examples

```
require(TargetSearchData)

# load CDF files, convert them and save them in a temporary directory
dest <- tempdir()
cdf <- tsd_cdffiles()[1:3]
cdf <- sapply(cdf, function(x) ncdf4_convert(x, outFile=file.path(dest, basename(x))))

# plot single m/z on many cdf files
ncdf4_plot_peak(cdf, massValues=c(144), timeRange=c(255, 265))
# add some style (note the usage of `slty` and `scol`)
ncdf4_plot_peak(cdf, massValues=c(144), timeRange=c(255, 265), scol=1:3, slty=1:3,
                main='Example Plot')

# plot many m/z on single cdf file (colors are set by matlines)
ncdf4_plot_peak(cdf[1], massValues=c(144, 145, 100, 218), timeRange=c(255, 265))
# add some styling (here simply use `lty` and `col`)
ncdf4_plot_peak(cdf[1], massValues=c(144, 145, 100, 218), timeRange=c(255, 265),
               col=1:4, lty=1, main='Example Plot')

# multiple m/z and files is possible but it gets messy
ncdf4_plot_peak(cdf, massValues=c(144, 145, 218), timeRange=c(255, 265),
               scol=1:3, main='Example Plot')

# do not plot anything and just get the data
z <- ncdf4_plot_peak(cdf, massValues=c(144, 145, 218), timeRange=c(255, 265), plot=FALSE)
str(z)

# using a tsSample object can achieve the same results
smp <- ImportSamplesFromDir(dest)
ncdf4_plot_peak(smp, massValues=c(144), timeRange=c(255, 265))
```

`ncdf4_update_ri` *Update retention time index on a NCDF4 file*

Description

Performs retention time index (RI) correction on a CDF file, using the retention markers found by `RIcorrect()`, or to force the markers time if, for example, the RI markers were not co-injected with the biological samples. It wraps around `rt2ri()`

Usage

```
ncdf4_update_ri(cdfFile, observed, standard)
```

Arguments

`cdfFile`	Path to the CDF file
`observed`	The observed RI markers retention times'.
`standard`	The RI of said markers.

Details

This function is similar to `fixRI()`, with the difference that is acts upon a single file, whereas `fixRI()` requires a tsSample object.

Value

Returns `invisible`

Note

This function is meant to be used internally. It is exposed for convenience.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`fixRI`, `RIcorrect`, `ImportFameSettings`

Examples

```
# pull a CDF file from package TargetSearchData as example
require(TargetSearchData)
cdffile <- tsd_cdffiles()[9]

# we need to convert from NCDF3 to NCDF4 in this example
# (NCDF3 files do not work)
nc4file <- ncdf4_convert(cdffile, 'example.nc4')

# to update the RI we need to pass two equal-length vectors of
# observed RT and standard RI values.
obsRT <- c(252, 311, 369) # approx time of RI markers
stdRI <- c(262320, 323120, 381020) # the exact RI of the markers

# finally update the RI for these file
ncdf4_update_ri(nc4file, obsRT, stdRI)
```

`NetCDFPeakFinding` *Peak picking algorithm from CDF files*

Description

This function reads a netcdf chromatogram file, finds the apex intensities and returns a list containing the retention time and the intensity matrices.

Usage

```
NetCDFPeakFinding(ncdf, massRange = NULL, Window = 15, IntThreshold = 10,
                  pp.method = "ppc", baseline = FALSE, ...)
```

Arguments

`ncdf`	A character string naming a netcdf file or a list generated by the function `peakCDFextraction`.
`massRange`	Deprecated. It is completely ignored but it is kept for compatibility with old scripts.
`Window`	The window used by peak picking method. The number of points actually used is `2*Window + 1`.
`IntThreshold`	Apex intensities lower than this value will be removed from the RI files.
`pp.method`	The pick picking method to be used. Options are `"smoothing"`, `"gaussian"` and `"ppc"`.
`baseline`	Logical. Should baseline correction be performed?
`...`	Extra options passed to `baseline`.

Details

The parameter `ncdf` should be either a path to a NetCDF file or a list generated by the function `peakCDFextraction`. This list contains elements named "Time", "Peaks", "massRange", "Index", and "baselineCorrected". Refer to the function's man page for details.

Formerly, the `massRange` parameter was a numeric vector with two components: lower and higher masses. Now, the mass range is detected automatically and it has no effect if it is set. It is kept only for compatibility reasons.

There are three peak picking algorithms that can be used. The `"smoothing"` method smooths the m/z curves by a moving average and then looks for a change of sign of the intensity difference between two consecutive points. The `"gaussian"` is exactly the same, but it uses a gaussian smoother instead of a moving average.

The `"ppc"` uses a sliding window and looks for the local maxima. This method is based on the R-package `ppc` (now unavailable).

Value

A three component list.

`Time`	The retention time vector.
`Peaks`	The intensity matrix. Rows are the retention times and columns are masses. The first column is the lower mass value and the last one is the higher mass.
`massRange`	The mass range.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`peakCDFextraction`

Examples

```
require(TargetSearchData)
# pick first CDF file
CDFfile <- tsd_cdffiles()[1]

# extract peaks of first chromatogram
peaks.1 <- NetCDFPeakFinding(CDFfile, Window = 15, IntThreshold = 10, pp.method = "ppc")
# scan acquisition times
head(peaks.1$Time)
# peaks in matrix form. first column is mass 85, last one is mass 320.
head(peaks.1$Peaks)
```

---

peakCDFextraction *NetCDF to R*

Description

This function reads a netcdf chromatogram (format 3 or 4) file and returns a list containing raw data (retention time, intensity matrix) for easy manipulation.

Usage

```
peakCDFextraction(cdfFile, massRange)
```

Arguments

cdfFile	A character string naming a netcdf file.
massRange	Deprecated. The mass range is extracted automatically.

Details

The `cdfFile` is a path to a CDF-3 or a CDF-4 format file. The CDF-4 format file is a custom file used on `TargetSearch`.

The `massRange` parameter is deprecated but kept for compatibility. The actual m/z range is detected automatically and returned to the user.

Value

A list with the following components:

Time	The retention time vector.
Index	The retention time index vector or NULL if unavailable.
Peaks	The intensity matrix. Rows are the retention times and columns are masses. The first column is the lower mass value and the last one is the higher mass.
massRange	The mass range.
baselineCorrected	Logical. It is `TRUE` if the file was baseline corrected by the function `baseline`. See note below.

Note

This function does not look for peaks, just extracts all the raw intensity values of the chromatogram file. Use `NetCDFPeakFinding` instead.

The function does not know whether a CDF file was baseline corrected (for example by the GC-MS software vendor), unless it was performed by the function `baseline`. It is perfectly possible to have a baseline corrected file and the element `baselineCorrected` be `FALSE`.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ncdf4_data_extract` for an updated and more flexible version of this function, `NetCDFPeakFinding`, `baseline`.

Examples

```
### take a NCDF3 file from the package TargetSearchData
require(TargetSearchData)
# pick a CDF file and extract its data
cdffile <- tsd_cdffiles()[4]
peakData <- peakCDFextraction(cdffile)

# it also works for NCDF4 files (we need to convert the file first)
nc4file <- ncdf4_convert(cdffile, tempfile(fileext='.nc4'))
peakData2 <- peakCDFextraction(nc4file)

# clean-up
unlink(nc4file)
```

`peakFind` *Intensities and RI matrices*

Description

This function returns a list of the intensities and RI matrices that were searched.

Usage

```
peakFind(samples, Lib, cor_RI, columns = NULL, showProgressBar = FALSE)
```

Arguments

`samples`	A `tsSample` object created by `ImportSamples` function.
`Lib`	A `tsLib` object created by `ImportLibrary` function with corrected RI values. See `medianRILib`.
`cor_RI`	A matrix of correlating selective masses RI for every sample. See `sampleRI`.
`columns`	Either `NULL`, a `character` vector, or an `integer` vector. In most cases, leave it as `NULL`. This parameter is used to configure the column names or positions of RI text files. See the documentation on the `text2bin` function for further details.

`showProgressBar`	Logical. Should the progress bar be displayed?

Value

A `tsMSdata` object.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ImportSamples`, `ImportLibrary`, `medianRILib`, `sampleRI`, `tsMSdata`, `tsLib`, `tsSample`

Examples

```
require(TargetSearchData)
data(TSExample)

# get RI file path
RI.path <- tsd_data_path()
refLibrary <- ImportLibrary(tsd_file_path("library.txt"))

# update RI file path
RIpath(sampleDescription) <- RI.path

peakData <- peakFind(sampleDescription, refLibrary, corRI)
# show peak Intensities.
head(Intensity(peakData), 2)

# How to get intensities for a particular metabolite
# just select the identifier. Here extract the intensities
# for the first metabolite in the library
IntMatrix <- Intensity(peakData)[[1]]
```

`plotFAME` *Plot a standard marker*

Description

Plots a given standard marker.

Usage

```
plotFAME(samples, RImatrix, whichFAME)
```

Arguments

`samples`	A `tsSample` object created by `ImportSamples` function.
`RImatrix`	A retention time matrix of the found retention time markers.
`whichFAME`	The retention marker to plot. Must be a number between 1 and the number of markers.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

RIcorrect, FAMEoutliers, tsSample

Examples

```
data(TSExample)
# plot Retention index standards 1 to 3
plotFAME(sampleDescription, RImatrix, 1)
plotFAME(sampleDescription, RImatrix, 2)
plotFAME(sampleDescription, RImatrix, 3)
```

---

plotPeak *Plot peaks*

Description

Plot a peak of a given metabolite in a given sample showing the search windows.

Usage

```
plotPeak(samples, Lib, metProf, rawpeaks, which.smp=1, which.met=1,
 massRange=NULL, corMass=FALSE)
```

Arguments

samples	A tsSample object created by ImportSamples function.
Lib	A tsLib object created by ImportLibrary function.
metProf	A metabolite profile object. See Profile
rawpeaks	A three component list containing the retention time, the intensity matrix, and the mass range. See peakCDFextraction.
which.smp	A numeric value indicating the sample.
which.met	A numeric value indicating the metabolite.
massRange	A two component numeric vector with the scan mass range to extract. Use NULL for automatic detection.
corMass	Logical. If TRUE, show only correlating masses for the selected metabolite. Show all masses otherwise.

Value

The output is the same as the function peakCDFextraction, that is, a list containing the retention time (Time), the intensity matrix (Peaks), the retention index (Index), the m/z range (massRange), and a flag indicating whether the chromatogram was baseline corrected (baselineCorrected).

The list can be recycled as the rawpeaks parameter for further plots (for example in a loop), so the CDF file doesn't need to be read again.

Note

This function was completely rewritten. For the old function, see `plotPeakSimple`.

In case the RI files are in text format and their column names are not standard (for example, when the files were generated with another software), use the global option `'TS_RI_columns'` or transform the RI files to `TargetSearch` binary format. See the documentation in function `text2bin`.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`plotPeakSimple`, `RIcorrect`, `tsMSdata`, `tsRim`, `peakCDFextraction`, `matplot`

Examples

```
require(TargetSearchData)
data(TSExample)

# update CDF and RI paths
CDFpath(sampleDescription) <- tsd_data_path()
RIpath(sampleDescription) <- tsd_data_path()

# Plot the peak "Valine" for sample number 1
grep("Valine", libName(refLibrary)) # answer: 3

# plot peak from the cdf file. The rawpeaks object can be recycled in order to plot
# other metabolites.
rawpeaks <- plotPeak(sampleDescription, refLibrary, metabProfile, which.smp=1,
        which.met=3, massRange=c(85,500), corMass=FALSE)
```

`plotPeakRI` *Plot peak RI across samples*

Description

Plot peak RI of the quant mass of a given metabolite across samples. This function can be used to visualize the elution time of a metabolite in the RI and RT dimension, and in combination with the function `sampleRI` for fine tuning.

Usage

```
plotPeakRI(samples, Lib, libID, dev=NULL, mz=NULL, RI=NULL, RT=NULL,
            method=c('RI', 'Intensity'), useRI=TRUE, main=NULL,
            col=NULL, int_range=c(2,6), cex_range=c(.7,6), key_width=2)
```

Arguments

`samples`	A `tsSample` object created by `ImportSamples` function.
`Lib`	A `tsLib` object created by `ImportLibrary` function.
`libID`	An index (integer or character) value representing the respective metabolite in the reference library `Lib`.

`dev`	The allowed retention index (RI) deviation or `NULL`.
`mz`	A list of m/z values to search or `NULL`.
`RI`	The expected retention index or `NULL`.
`RT`	The expected retention time to search for instead of retention index, or `NULL`.
`method`	A character vector used to decided what peak should be chosen in case there are ambiguous peaks. If `'RI'`, then the closest peak to the expected RI (or RT) is chosen. If `'Intensity'`, then the highest is taken.
`useRI`	Logical. Should the RI or RT be displayed in the y-axis?
`main`	The title for the plot. If `NULL`, then the metabolite name is displayed.
`col`	A color vector (length > 2) to show different levels of peak intensity.
`int_range`	A length-two vector. The limits of the intensity for the color key. Note that the intensity is expressed in log10, for example, a value of 2 represents an intensity of 100.
`cex_range`	The 'cex' range value of the points. Lower-intense peaks are represented as points of smaller size.
`key_width`	The width in cm of the area allocated for the color key.

Details

This function uses internally `FindAllPeaks`, so the same rules apply, i.e., the parameters `dev`, `mz`, `RI`, and `RT` have preference over the settings of the metabolite indexed by `libID`.

In the plot, the x-axis are samples as defined by the object `samples`. The y-axis is retention index (RI) as shown on the left-hand-size. On the right-hand-size y-axis the approximate retention time (RT) is shown. This is because the RT varies across samples, therefore it is averaged for displays purposes. If `useRI==FALSE`, then the RT is displayed on the left hand size and the RI is averaged and shown on the left.

Note that in general, the RI of the library or the RI given by the user is used for peak searching, regardless of the value of parameter `useRI`. To search for RT instead, then the parameter `RT` is required, as well as the deviation parameter `dev`, as in the function `FindAllPeaks`.

The point size is proportional to the log10 of the peak intensity. Their size is controlled by the parameters `int_range` and `cex_range`. By default, intensities of 100 (log10 => 2) or lower are shown with `cex=0.7`, while intensities greater than 1000000 (log10 => 6) as displayed with `cex=6`. This also affects the scaling of the color key.

The best peaks, selected according to `method`, are shown with a black border, while the other are shown with no border and slightly transparent.

The output is the RI of the best peaks or invisible. Note that if no peak is found, then no plot is drawn.

Value

Returns invisible or a numeric vector with the corresponding RI of the best peak chosen by `method`. The RI is returned even when the parameters `useRI` is set to `TRUE` or the parameter `RT` is specified.

For a more comprehensive output (but less nicer plot), check the function `ri_plot_peak`.

Author(s)

Alvaro Cuadros-Inostroza

See Also

The function ri_plot_peak which is a low level version of this function, as well as FindAllPeaks for details of the peak searching parameters. Check also referenced functions sampleRI, ImportSamples, and ImportLibrary

Examples

```
def.par <- par(no.readonly = TRUE) # save parameters for resetting

# load pre-calculated example data files and objects
require(TargetSearchData)
data(TSExample)

# get and set the RI file path
RIpath(sampleDescription) <- tsd_data_path()

# search all peaks of Valine (GC.3) and selective masses. Retention index
ri <- plotPeakRI(sampleDescription, refLibrary, 'GC.3')

# increase deviation, change m/z to search, change colors and title
main <- 'Valine'
cols <- c('red', 'blue', 'green')
ri <- plotPeakRI(sampleDescription, refLibrary, 'GC.3', dev=4000, mz=144,
                 main=main, col=cols)

# plot by RT instead. Note the RI is still returned and used for searching
ri <- plotPeakRI(sampleDescription, refLibrary, 'GC.3', useRI=FALSE)

# same, but search by RT instead of RI. Note that the deviation 'dev' is
# required
ri2 <- plotPeakRI(sampleDescription, refLibrary, 'GC.3', useRI=FALSE, RT=261, dev=2)

stopifnot(ri == ri2) # should be the same

par(def.par) # reset to default
```

---

plotPeakSimple *Plot peaks - simple (old) interface*

Description

Plot selected ions in a given time range. This is the old plot peak interface. *Note*: This function is considered **deprecated** and its usage is discouraged; users should use ncdf4_plot_peak.

Usage

```
plotPeakSimple(rawpeaks, time.range, masses, cdfFile = NULL, useRI = FALSE,
  rimTime = NULL, standard = NULL, massRange = NULL, ...)
```

Arguments

rawpeaks	A three component list containing the retention time, the intensity matrix, and the mass range. See peakCDFextraction.

`time.range`	The time range to plot in retention time or retention time index units to plot.
`masses`	A vector containing the ions or masses to plot.
`cdfFile`	The name of a CDF file. If a file name is specified, the ions will be extracted from there instead of using `rawpeaks`.
`useRI`	Logical. Whether to use Retention Time Indices or not.
`rimTime`	A retention time matrix of the found retention time markers. It is only used when `useRI` is `TRUE`.
`standard`	A numeric vector with RI values of retention time markers. It is only used when `useRI` is `TRUE`.
`massRange`	A two component numeric vector with the scan mass range to extract or `NULL` for automatic detection.
`...`	Further options passed to `matplot`.

Note

This function used to be named 'plotPeak'. This function was completely rewritten so we kept the old version and renamed it 'plotPeakSimple'.

Important: This function is considered **deprecated**, though it is not yet labeled as such. Please consider using `ncdf4_plot_peak`. This function, however, needs CDF files version 4.

That said, this function can still be used to plot peaks for CDF files version 3 without the need to convert them to CDF format 4.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`plotPeak`, `RIcorrect`, `tsMSdata`, `tsRim`, `peakCDFextraction`, `matplot`, `ncdf4_plot_peak` for the new interface.

Examples

```
require(TargetSearchData)
data(TSExample)

# update CDF path
CDFpath(sampleDescription) <- tsd_data_path()

# Plot the peak "Valine" for sample number 1
grep("Valine", libName(refLibrary)) # answer: 3
# select the first file
cdfFile  <- CDFfiles(sampleDescription)[1]

# select "Valine" top masses
top.masses <- topMass(refLibrary)[[3]]

# plot peak from the cdf file
plotPeakSimple(cdfFile = cdfFile, time.range = libRI(refLibrary)[3] + c(-2000,2000),
    masses = top.masses, useRI = TRUE, rimTime = RImatrix[,1],
    standard = rimStandard(rimLimits), massRange = c(85, 500))
```

```
# the same, but extracting the peaks into a list first. This may be better if
# you intend to loop through several peaks.
rawpeaks <- peakCDFextraction(cdfFile, massRange = c(85,500))
plotPeakSimple(rawpeaks, time.range = libRI(refLibrary)[3] + c(-2000,2000),
    masses = top.masses, useRI = TRUE, rimTime = RImatrix[,1],
    standard = rimStandard(rimLimits), massRange = c(85, 500))
```

`plotRefSpectra` *Plot reference spectrum of a compound*

Description

A simple function to plot the reference spectrum of a compound in the style of common visualization tools. This is useful when comparing the spectrum of a metabolite with such software tools.

Usage

```
plotRefSpectra(lib, libID, col='darkblue', main=NULL, xlab='m/z', ylab='Intensity',
                 label=TRUE, cex=0.8, label_col=NULL, sel_font=4, type='h', ...)
```

Arguments

`lib`	A `tsLib` object created by `ImportLibrary` function.
`libID`	An index (integer or character) value representing the respective metabolite in the reference library `lib`. If missing, the first metabolite in the library is taken.
`col`	The color(s) of the vertical lines. This option is passed to `plot`.
`main`	The title for the plot. If `NULL`, then the metabolite name is displayed.
`xlab`	The *x* axis label. Passed to `plot`.
`ylab`	The *y* axis label. Passed to `plot`.
`label`	Logical. It controls whether the *m/z* labels are displayed.
`cex`	The `cex` values (numerical vector) used to scale the *m/z* labels of the plot.
`label_col`	The color vector for the *m/z* labels.
`sel_font`	The font for the *m/z* labels of the *selective* masses. This should be an integer like the parameter `font` documented in `par`.
`type`	A character that specifies the plot type. Defaults to `'h'` (for vertical bars) and **cannot** be changed by the user (it has no effect).
`...`	Extra graphical parameters passed to `plot`.

Details

This is a simple function to plot the spectrum of a reference metabolite. It is expected that the spectrum for that metabolite is not empty, in which case nothing is plotted and a warning is thrown.

The *m/z* label positions are determined automatically such that they do not intersect with each other and do not touch the vertical lines of the plot.

Value

Returns `invisible()`.

Author(s)

Alvaro Cuadros-Inostroza

See Also

See the function `ImportLibrary` for importing a metabolite library. See `plot` and `par` for graphical parameters.

Examples

```
def.par <- par(no.readonly = TRUE) # save parameters for resetting

# load pre-calculated example data files and objects
data(TSExample)

# plot the reference spectrum of the compound ID 'GC.3'
plotRefSpectra(refLibrary, 'GC.3')

# that is equivalent to
plotRefSpectra(refLibrary['GC.3'])

# change some graphical parameter and add a grid via 'panel.first'
plotRefSpectra(refLibrary, 'GC.3', col='darkred', main='Valine', xlab='mz',
               ylab='intensity', cex=0.7, label_col='blue', sel_font=2,
               panel.first=grid())

par(def.par) # reset to default
```

`plotRIdev` *Plot Retention Time Index Deviation*

Description

`plotRIdev` plots the Retention Time Index Deviation of a given set of metabolites. `plotAllRIdev` saves the plots of the RI deviations of all the metabolites in the library object into a PDF file.

Usage

```
plotRIdev(Lib, peaks, libID = 1, ...)

plotAllRIdev(Lib, peaks, pdfFile, width = 8, height = 8, ...)
```

Arguments

`Lib`	A `tsLib` object created by `ImportLibrary` function.
`peaks`	A `tsMSdata` object. See `peakFind`.
`libID`	A numeric vector providing the indices of the metabolites to plot.
`pdfFile`	A file name where the plot will be saved. Only `plotAllRIdev`.
`width, height`	The width and height of the plots in inches. Only `plotAllRIdev`.
`...`	Further options passed to `pdf`.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ImportLibrary`, `tsLib`, `tsMSdata`, `pdf`

Examples

```
require(TargetSearchData)
data(TSExample)

# get RI file path
RI.path <- tsd_data_path()
# update RI file path
RIpath(sampleDescription) <- RI.path

peakData <- peakFind(sampleDescription, refLibrary, corRI)

# Plot RI deviation of metabolite "Valine"
grep("Valine", libName(refLibrary)) # answer: 3
plotRIdev(refLibrary, peakData, libID = 3)

# Plot an RI deviation overview of the first nine metabolites
plotRIdev(refLibrary, peakData, libID = 1:9)

# Save all RI deviation into a pdf file
plotAllRIdev(refLibrary, peakData, pdfFile = "RIdeviations.pdf")
```

`plotSpectra` *Plot a Spectra Comparison*

Description

`plotSpectra` plots a contrast between the reference spectra and the median spectra of a given metabolite in the library. `plotAllRIdev` saves the plots of the median-reference spectra comparisons of all the metabolites in the reference library into a PDF file.

Usage

```
plotSpectra(Lib, peaks, libID = 1, type = c("ht", "ss", "diff"))

plotAllSpectra(Lib, peaks, type = "ht", pdfFile, width = 8, height = 8, ...)
```

Arguments

`Lib`	A `tsLib` object created by `ImportLibrary` function.
`peaks`	A `tsMSdata` object. See `peakFind`.
`libID`	A numeric vector providing the indices of the metabolites to plot.
`type`	The type of the plot. Options are `"ht"`, head-tail plot, `"ss"`, side by side plot, and `"diff"`, spectrum difference plot.

`pdfFile`	A file name where the plot will be saved. Only `plotAllRIdev`.
`width, height`	The width and height of the plots in inches. Only `plotAllRIdev`.
`...`	Further options passed to `pdf`.

Details

The median spectra is obtained by computing the median intensity of every ion across the samples. The median and the reference spectra values are scaled to vary between 0 and 999 in order to make them comparable.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`tsLib`, `tsMSdata`,`pdf`

Examples

```
require(TargetSearchData)
data(TSExample)

# get RI file path
RI.path <- tsd_data_path()
# update RI file path
RIpath(sampleDescription) <- RI.path

peakData <- peakFind(sampleDescription, refLibrary, corRI)

# Plot a comparison RI deviation of metabolite "Valine"
grep("Valine", libName(refLibrary)) # answer: 3
plotSpectra(refLibrary, peakData, libID = 3, type = "ht")

# Plot the spectra "side by side"
plotSpectra(refLibrary, peakData, libID = 3, type = "ss")

# Plot the spectra difference
plotSpectra(refLibrary, peakData, libID = 3, type = "diff")
```

`Profile` *Average the correlating masses for each metabolite*

Description

This function makes a profile from the masses that correlate for each metabolite.

Usage

```
Profile(samples, Lib, peakData, r_thres = 0.95, method = "dayNorm", minPairObs = 5)
```

Arguments

`samples`	A `tsSample` object created by `ImportSamples` function.
`Lib`	A `tsLib` object created by `ImportLibrary` function with corrected RI values. See `medianRILib`.
`peakData`	A `tsMSdata` object. See `peakFind`.
`r_thres`	A correlation threshold.
`method`	Normalisation method. Options are `"dayNorm"`, a day based median normalisation, `"medianNorm"`, normalisation using the median of all the intensities of a given mass, and `"none"`, no normalisation at all.
`minPairObs`	Minimum number of pair observations. Correlations between two variables are computed using all complete pairs of observations in those variables. If the number of observations is too small, you may get high correlations values just by chance, so this parameters is used to avoid that. Cannot be set lower than 5.

Value

A `tsProfile` object. The slots are:

`Info`	A data frame with a profile of all masses that correlate.
`Intensity`	A list containing peak-intensity matrices, one matrix per metabolite.
`RI`	A list containing RI matrices, one matrix per metabolite.
`profInt`	A matrix with the averaged intensities of the correlating masses.
`profRI`	A matrix with the averaged RI of the correlating masses.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ImportSamples`, `ImportLibrary`, `medianRILib`, `peakFind`, `tsProfile`

Examples

```
require(TargetSearchData)
data(TSExample)

# get RI file path
RI.path <- tsd_data_path()
# update RI file path
RIpath(sampleDescription) <- RI.path
# Import Library
refLibrary       <- ImportLibrary(tsd_file_path('library.txt'))
# update median RI
refLibrary       <- medianRILib(sampleDescription, refLibrary)
# get the sample RI
corRI            <- sampleRI(sampleDescription, refLibrary, r_thres = 0.95)
# obtain the peak Intensities of all the masses in the library
peakData         <- peakFind(sampleDescription, refLibrary, corRI)
# make a profile of the metabolite data
metabProfile     <- Profile(sampleDescription, refLibrary, peakData, r_thres = 0.95)
```

```
# same as above, but with different thresholds.
metabProfile      <- Profile(sampleDescription, refLibrary, peakData,
                        r_thres = 0.9, minPairObs = 5)
```

`ProfileCleanUp` *Reduce redundancy of the profile*

Description

This function reduces/removes redundancy in a profile.

Usage

```
ProfileCleanUp(Profile, timeSplit=500, r_thres=0.95, minPairObs=5,
    prioritization=c('mass','score'), corMass=1, score=0,
    show=c('unidentified','knowns','full'))
```

Arguments

`Profile`	A `tsProfile` object. See `Profile`.
`timeSplit`	A RI window.
`r_thres`	A correlation threshold.
`minPairObs`	Minimum number of pair observations. Correlations between two variables are computed using all complete pairs of observations in those variables. If the number of observations is too small, you may get high correlations values just by chance, so this parameters is used to avoid that. Cannot be set lower than 5.
`prioritization`	Selects whether the metabolite suggestion should be based on the number of correlation masses (`mass`) or the score (`score`).
`corMass`	Metabolites with a number of correlation masses lower than `score` will be marked as 'Unidentified RI'
`score`	Metabolites with a score lower than `score` will be marked as unidentified.
`show`	A character vector. If `unidentified`, all non-redundant metabolites will be returned; if `knowns`, only returns those metabolites with correlation masses and score greater than the given values; and if `full`, it shows all redundant metabolites, which may be useful to retrieve the data from misidentified metabolites.

Details

Metabolites that are inside a `timeSplit` window will be correlated to see whether the metabolites are potentially the same or not, by using `r_thres` as a cutoff. If so, the best candidate will be chosen according to the value of `prioritization`: If 'mass', then metabolites will be suggested based on number of correlating masses, and if 'score', then the score will be used. Metabolites that don't have al least `corMass` correlating masses and `score` score will be marked as 'unidentified' and not will be suggested, unless all the metabolites in group are unidentified.

For example, suppose that three metabolites A (CM=3, S=900), B (CM=6, S=700), C (CM=5, S=800) correlate within the same time group, where CM is the number of correlating masses and S is the score.

- If prioritization='mass', corMass=3, score=650, then the suggested order is B, C, A.

- If prioritization='mass', corMass=3, score=750, then the suggested order is C, A, B.
- If prioritization='mass', corMass=3, score=850, then the suggested order is A, B, C.
- If prioritization='score', corMass=3, score=650, then the suggested order is A, C, B.
- If prioritization='score', corMass=4, score=650, then the suggested order is C, B, A.
- If prioritization='score', corMass=4, score=850, then the suggested order is C, A, B.

Note that by choosing `prioritization='mass'`, `score=0`, and `corMass=1` you will get the former behavior (`TargetSearch` <= `1.6`).

Value

A `tsProfile` object with a non-redundant profile of the masses that were searched and correlated, and intensity and RI matrices of the correlating masses.

`slot "Info"`	A data frame with a profile of all masses that correlate and the metabolites that correlate in a `timeSplit` window.
`slot "profInt"`	A matrix with the averaged intensities of the correlating masses.
`slot "profRI"`	A matrix with the averaged RI of the correlating masses.
`slot "Intensity"`	A list containing peak-intensity matrices, one matrix per metabolite.
`slot "RI"`	A list containing RI matrices, one matrix per metabolite.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`Profile`, `tsProfile`

Examples

```
# load example data
require(TargetSearchData)
data(TSExample)

RI.path <- tsd_data_path()
refLibrary <- ImportLibrary(tsd_file_path("library.txt"))
# update RI file path
RIpath(sampleDescription) <- RI.path
# Import Library
refLibrary        <- ImportLibrary(tsd_file_path('library.txt'))
# update median RI
refLibrary        <- medianRILib(sampleDescription, refLibrary)
# get the sample RI
corRI             <- sampleRI(sampleDescription, refLibrary, r_thres = 0.95)
# obtain the peak Intensities of all the masses in the library
peakData          <- peakFind(sampleDescription, refLibrary, corRI)
metabProfile      <- Profile(sampleDescription, refLibrary, peakData, r_thres = 0.95)

# here we use the metabProfile previously calculated and return a "cleaned" profile.
metabProfile.clean <- ProfileCleanUp(metabProfile, timeSplit = 500,
                      r_thres = 0.95)
```

```
# Different cutoffs could be specified
metabProfile.clean <- ProfileCleanUp(metabProfile, timeSplit = 1000,
                        r_thres = 0.9)
```

quantMatrix *Create an intensity matrix using quantification masses*

Description

Create an intensity matrix using quantification masses. The quantification masses can be specified when importing the library file or automatically.

Usage

```
quantMatrix(Lib, metabProfile, value=c("quantmass", "maxint", "maxobs"), selmass=FALSE)
```

Arguments

`Lib`	A `tsLib` object created by `ImportLibrary` function.
`metabProfile`	A `tsProfile` object. The final result of the package. This object is generated by either `Profile` or `ProfileCleanUp`.
`value`	The method to select automatically the quantification mass. The options are: “maxint” which selects the selective mass with the highest median intensity across all samples; “maxobs” which selects the mass which the most observations (the one with fewer missing values; or “quantmass” which uses the selected quantification mass defined by the library
`selmass`	Logical. If `TRUE`, then only selective masses are considered if the option `value` is either “maxint” or “maxobs”, otherwise all masses are considered (the default behavior). If `value` is “quantmass”, this argument has no effect

Value

An intensity matrix with metabolites as rows and samples as columns. The column names are the sample names of the respective `tsSample` object and the rownames correspond with the library unique identifiers.

In addition, the matrix has these attributes:

- “quantMass” is a numeric vector that contains the quantification masses that was selected.
- “isSelMass” is a logical vector that indicates whether the quantification mass is also a selected mass.
- “isCorMass” to indicate that the mass is one of the correlating masses.
- “libNames” the metabolite names (a character vector).

Author(s)

Alvaro Cuadros-Inostroza

See Also

`tsLib`, `tsMSdata`

Examples

```
require(TargetSearchData)
data(TSExample)

# process chromatograms and get a profile
RI.path <- tsd_data_path()
RIpath(sampleDescription) <- RI.path
refLibrary    <- ImportLibrary(tsd_file_path('library.txt'))
refLibrary    <- medianRILib(sampleDescription, refLibrary)
corRI         <- sampleRI(sampleDescription, refLibrary, r_thres = 0.95)
peakData      <- peakFind(sampleDescription, refLibrary, corRI)
metabProfile  <- Profile(sampleDescription, refLibrary, peakData, r_thres = 0.95)

# get a Matrix using use default values, ie, use the quantification masses
# defined in the library
quantMat <- quantMatrix(refLibrary, metabProfile)
quantMat

# use 'maxint' to select the metabolites with the highest median intensity
quantMat <- quantMatrix(refLibrary, metabProfile, 'maxint')

# use 'maxobs' to select the metabolites with the most observed values
quantMat <- quantMatrix(refLibrary, metabProfile, 'maxobs')
```

---

ri2rt *Retention Time Index to Retention Time conversion*

Description

Convert retention time indices to retention times indices based on observed FAME RI and their standard values.

Usage

```
ri2rt(riTime, rt.observed, ri.standard)
```

Arguments

`riTime`	And RI vector or matrix to convert to Retention Time.
`rt.observed`	The observed FAME RT's. It could be a vector or a matrix.
`ri.standard`	The standard RI for each FAME

Details

This function is the inverse of `rt2ri`.

Value

The converted RT

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`RIcorrect`, `FAMEoutliers`

Examples

```
# RI standards
standard <- c(100, 200, 300, 400, 500)
# observed standard retention times
observed <- c(10.4, 19.3, 32.4, 40.2, 50.3)
# a random set of retention times
RI      <- runif(100,90,600)
# the corrected RIs
RT      <- ri2rt(RI, observed, standard)
```

`RIcorrect` *Peak picking from CDF files and RI correction*

Description

This function reads from CDF files, finds the apex intensities, converts the retention time to retention time index (RI), and writes RI corrected text files (a.k.a. RI files). In addition, it can perform baseline correction and also convert files to the new NetCDF-4 TargetSearch format.

Usage

```
RIcorrect(samples, rimLimits = NULL, massRange=NULL, Window, IntThreshold,
          pp.method="ppc", showProgressBar=FALSE, baseline=FALSE,
          writeCDF4path=TRUE, ...)
```

Arguments

`samples`	A `tsSample` object usually created by `ImportSamples` function.
`rimLimits`	A `tsRim` object. If set to `NULL`, no retention time will be performed. See `ImportFameSettings`.
`massRange`	Deprecated. It is completely ignored but it is kept for compatibility with old scripts.
`Window`	The window used for smoothing. The number of points actually used is `2*Window + 1`. It must be an integer. See details.
`IntThreshold`	Apex intensities lower than this value will be removed from the RI files.
`pp.method`	Peak picking method. Options are "smoothing", "gaussian" and "ppc". See details.
`showProgressBar`	Logical. Should the progress bar be displayed?
`baseline`	Logical. Should baseline correction be performed?
`writeCDF4path`	Whether or not convert a CDF-3 into a CDF-4. It can take a logical value or a character vector representing file paths. See details below.
`...`	A list of options passed to `baseline`.

Details

There are three pick picking methods available: "ppc", "smoothing", "gaussian".

The "ppc" method (default) implements the peak detection method described in the ppc package. It looks for the local maxima within a `2*Window + 1` scans for every mass trace.

The "smoothing" method calculates a moving average of `2*Window + 1` points for every mass trace. Then it looks for a change of sign (from positive to negative) of the difference between two consecutive points. Those points will be returned as detected peaks.

The "gaussian" method behaves similar to the "smoothing" method, but instead a gaussian smoother is used instead of the moving average.

To work out a suitable `Window` value, the following might be useful: `Window = (SR * PW - 1) / 2`, where SR is the scan rate of the MS instrument and PW is the peak width. Because `Window` is an integer, the resulting value must be rounded. For example, for SR = 20 scans per second, a PW = 1.5 seconds, then `Window` = 14.5, which can be rounded to 15.

The RI file type is determined by the output of `fileFormat` method applied to the `tsSample` input object. To choose between the available formats ("binary" and "text"), select it with `fileFormat` method before calling `RIcorrect`.

The parameter `writeCDF4path` is used to convert CDF-3 files into a custom CDF-4 format. It can be logical or a character vector. If `TRUE`, the default, the CDF-3 files will be converted automatically to CDF-4 format. The files will be saved in the same directory as the original CDF-3. If it is character vector representing file paths, then the CDF-4 will be saved in those paths instead (re-cycled to the length of samples). Finally, if `FALSE`, then no CDF conversion will be performed. This is not recommended, but if possible if you want to match the old `TargetSearch` behaviour. Note that also the baseline correction or the retention time indices will not be updated as well.

If `baseline` is `TRUE`, the CDF files will be baseline corrected by passing the extra parameters to `baseline`. See there for details.

Value

A matrix of the retention time of the markers. Every column represents a sample, while rows are RT markers. In addition the matrix has two extra attributes named "intensity" and "mass". The former is a matrix of same dimensions which contain the peak intensity of the markers, while the latter represents the `m/z` value of the respective marker (this is taken from the `rimLimits` object).

Note

This function expects that the file extension of the input files is **not** `.nc4`, unless those files were created by `TargetSearch`. If this is the case, the function may throw an error, in particular if `writeCDF4path` is used. However, this should not be an issue as most vendor software export the chromatograms as `.cdf` files.

If the parameter `writeCDF4path` points to a new path, you may need to call the methods `ncdf4Convert` or `CDFpath` to update the path of the `tsSample` object, as this is not done automatically.

Likewise, if the files are `TargetSearch`'s, but do **not** have the `.nc4` extension, this function will create new `.nc4` files.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

The functions ImportSamples, ImportFameSettings, NetCDFPeakFinding, FAMEoutliers, the method ncdf4Convert, and the class definitions tsSample, tsRim.

Examples

```
require(TargetSearchData)
# import refLibrary, rimLimits and sampleDescription.
data(TSExample)
# get the CDF files
cdfpath <- tsd_data_path()
cdfpath
list.files(cdfpath)
# update the CDF path
CDFpath(sampleDescription) <- cdfpath

# change file format of RI files as bin
fileFormat(sampleDescription) <- 'binary'

# Parameters: Intensity Threshold = 50 peak detection method = "ppc", window = 15
# To match the old behavior, the do not create CDF-4 Files (not recommended)
RImatrix <- RIcorrect(sampleDescription, rimLimits, writeCDF4path=FALSE,
            Window = 15, pp.method = "ppc", IntThreshold = 50)

# Convert to CDF-4 (recommended) with same parameters
# Note: save files in same directory (as
RImatrix <- RIcorrect(sampleDescription, rimLimits, writeCDF4path=".",
            Window = 15, pp.method = "ppc", IntThreshold = 50)

# we need to update the sampleDescription to use the new files
# this is not done automatically
sampleDescription <- ncdf4Convert(sampleDescription, ".")

# you can try other parameters and other peak picking algorithm
RImatrix <- RIcorrect(sampleDescription, rimLimits,
            Window = 15, pp.method = "smoothing", IntThreshold = 10)

RImatrix <- RIcorrect(sampleDescription, rimLimits,
            Window = 15, pp.method = "ppc", IntThreshold = 100)

# the attributes intensity and mass can be accessed with by the attr function
intensities <- attr(RImatrix, "intensity")
mass <- attr(RImatrix, "mass")

# they should be identical
identical(mass, rimMass(rimLimits))
```

`riMatrix` *Retention Time Index Matrix*

Description

A function to search for retention index RI markers.

Usage

```
riMatrix(samples, rim)
```

Arguments

`samples`	A `tsSample` object created by `ImportSamples` function.
`rim`	A `tsRim` object. See `ImportFameSettings`.

Details

This function works similar to `RIcorrect`, but searches for RI markers in RI files (not in CDF files). Can be used to retrieve the retention times of RI markers in already processed files.

Note that it does not perform any RI adjustment. See `fixRI`.

Value

A matrix of the retention time of the markers. Every column represents a sample, while rows are RT markers. In addition the matrix has two extra attributes named "intensity" and "mass". The former is a matrix of same dimensions which contain the peak intensity of the markers, while the latter represents the `m/z` value of the respective marker (this is taken from the `rimLimits` object).

Note

In case the RI files are in text format and their column names are not standard (for example, when the files were generated with another software), use the global option `'TS_RI_columns'` or transform the RI files to `TargetSearch` binary format. See the documentation in function `text2bin`.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`RIcorrect`, `FAMEoutliers`,`ImportSamples`, `ImportFameSettings`, `fixRI`

Examples

```
require(TargetSearchData)
# import refLibrary, rimLimits and sampleDescription.
data(TSExample)
# get the CDF files
cdfpath <- tsd_data_path()

# select a subset of samples
smp <- sampleDescription[1:4]

# update the CDF path
CDFpath(smp) <- cdfpath

# make a copy of the RI markers object
rim <- rimLimits

# run RIcorrect
RImat <- RIcorrect(smp, rim, massRange = c(85,320), writeCDF4path=FALSE,
            Window = 15, pp.method = "ppc", IntThreshold = 50)

# extract the retention times of the markers
RImat2 <- riMatrix(smp, rim)

# both matrices should be equal
stopifnot( all.equal(RImat, RImat2, tolerance=1e-8) )
```

---

ri_data_extract *Extract peak data from a RI file*

Description

A convenient function to extract peak data from a RI file. This function is mostly intended for debugging purposes, for example, when a given peak is not found within a expected time range (which could be because a faulty retention time correction). The arguments of this function are similar as those of the `ncdf4_data_extract` function.

Usage

```
ri_data_extract(RIfile, massValues, timeRange, useRT = FALSE, ...)
```

Arguments

`RIfile`	A path to a RI file format (binary or text).
`massValues`	A numeric vector representing m/z values. The values will be converted to integers internally.
`timeRange`	A numeric vector of even length, or, a two-column matrix. This parameter represents the lower and upper time limits. See details below.
`useRT`	Logical. If `TRUE`, the time range is in seconds, otherwise if `FALSE` (default), the time range is in retention time index units (`TRUE`).
`...`	Extra parameters passed to internal functions. Currently, only the `columns` parameter is used for text-format RI files. This, however, should **not** be necessary. See also `text2bin`.

Details

The function takes a RI file generated by `TargetSearch` (binary or text works) and extracts all the peaks detected of given *m/z* ion traces in a given time range. The time range can be in seconds or arbitrary retention time index units.

The parameter `timeRange` can be either a numeric vector of even length or a two-column matrix. If it is a matrix, the first column represent the lower bounds while the second the upper ones. If it is a vector, it must have at least two elements, which in this case represent the lower and upper limits, or if longer, it will be coerced to matrix internally (**Important**: the values are filled by *columns*). In any case, the number of rows of the (coerced) matrix must divide the length of the `massValues` parameter, such that each row corresponds to a time range for each *m/z* value. This is done so the ranges can be recycled.

The output is simply a four column matrix in which a row is a peak of a given ion trace at a retention time (index). The output is always a matrix even if no peaks are found, in which case the number of rows is zero.

Value

A four column matrix with column names described below. Each row represents a peak in the specified time range. Potentially, there could be zero rows if no peaks are found.

- `RI` retention time index of the peak.
- `RT` retention time.
- `Intensity` peak height intensity
- `mz` the *m/z* of the peak

Note

This function is intended to be used internally or by advances users. It can be used for debugging when `TargetSearch` fails to find a peak.

See also the function `FindAllPeaks` which offers a similar functionality but with different input parameters.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`text2bin`, `ncdf4_data_extract`, `FindAllPeaks`, `FindPeaks`

Examples

```
require(TargetSearchData)
path <- tsd_data_path()
rifile <- tsd_rifiles()[1]

# extract peak data for m/z 116 and 117 in time range 180 - 220 seconds
data <- ri_data_extract(rifile, c(116, 117), c(180, 220), useRT=TRUE)

# same but using Retention Time Index
data <- ri_data_extract(rifile, c(116, 117), c(205000, 228000), useRT=FALSE)

# different time ranges for each m/z
trange <- c(180, 240, 280, 190, 250, 290)
data <- ri_data_extract(rifile, c(116, 144, 147), trange, useRT=TRUE)

# Note: the range definition above is equivalent to
trange <- cbind(c(180,240,280), c(190, 250, 290))
```

`ri_plot_peak` *Plot peak RI across samples - low level interface*

Description

Plot retention index or time of peaks from RI files or samples. This is a low level interface to `plotPeakRI` that not relies on a metabolite library. This function is useful for quick peak data visualization.

Usage

```
ri_plot_peak(obj, massValues, timeRange, useRT = TRUE, showRT = useRT, sizefun = NULL, plot = TRUE, .
```

Arguments

`obj`	Either a `tsSample` object or a character vector of paths to RI files.
`massValues`	A numeric vector representing *m/z* values.
`timeRange`	A numeric vector of length 2 representing the lower and upper time limits.
`useRT`	Logical. If `TRUE` (default), the time range is expressed in seconds, otherwise if `FALSE`, the time range is in retention time index units.
`showRT`	Logical. Whether the *x*-axis of the resulting plot should represent seconds (`TRUE`) or retention index units (`FALSE`). This allows, for example, to search for a time range in seconds and display a graph in index units.
`sizefun`	A function that could be used to scale the point size of the plot via the parameter 'cex', for example, to scale the peak intensity. This function should take a numeric vector as input and return a positive value. If `NULL`, then a default function will be used. This function is called using the peak intensity as input.
`plot`	Logical. If `FALSE` nothing is plotted. This option may be used to simply extract the data from the RI files, but see also `ri_data_extract`.
`...`	Extra arguments, such as `pch`, `col`, ..., to be passed to the plotting functions. See details below.

Details

The function uses internally `ri_data_extract` wrapped with a call to `lapply`. Because the output of `lapply` is a `list`, the list is transformed to a matrix by calling `rbind`. A column of sample indices called `sample` is added so each row can be traced back to a sample.

The *x*-axis of the plot are indexed samples (or RI files), i.e., first sample is 1, second 2, and so on. The *y*-axis is retention time (RT) or retention index (RI) depending on the value of `showRT`. Note that you can search by RT and plot by RI depending on the value of `useRT`.

Plot styling can be achieved by passing plotting extra parameters such as `col`, `pch`, `lty`, and others (see `par` and `plot`). Because the styles can be applied per sample or per *m/z* value, it is possible to prefix a 's' to those parameters to specify that the style applies to samples and no masses. The available parameters are `scol`, `spch`, `sbg` and `scex`. See examples and also the notes below.

The parameter `sizefun` is a function that takes a numeric vector as input and returns a positive value which correspond to the point size (parameter `cex`). This function will be applied on the peak intensities and the result will be used as `cex`. If `NULL`, the scaling function is implemented as `(log10(x/100) * (9/8) + (1/2)`, where `x` is the peak intensity. Note that if either `cex` or `scex` is passed to the function, then `sizefun` will have no effect.

The logical parameter `plot` controls whether a plot is drawn. This might be useful for simply extracting the data for custom plotting with other packages such as `ggplot2`.

Value

Returns `invisible` or a five-column matrix (see below), where the number of rows correspond to the number of peaks (or points) in the searched range. If no peaks are found in the given range, then it returns `NULL`.

- `RI` retention time index or the peak.

- `RT` retention time.
- `Intensity` peak height intensity.
- `mz` the *m/z* of the peak.
- `sample` an integer representing the sample index.

Compare this list with `ri_data_extract`.

Note

This function is intended for advanced users and for debugging purposes when `TargetSearch` fails to detect a peak.

Not all graphical parameters passed by `...` will work, in particular `panel.first` and `panel.last` are known to *not* work. Therefore, for more personalized plots, it is recommended to set `plot=FALSE` and use the returned data for plotting.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`plotPeakRI`, `ri_data_extract`

Examples

```
require(TargetSearchData)

# get RI files from TargetSearchData
path <- tsd_data_path()
rifiles <- tsd_rifiles()

# simple plot no style
z <- ri_plot_peak(rifiles, massValues=c(144, 145, 100, 218), timeRange=c(255, 265))
# watch output data
head(z)

# add some style
ri_plot_peak(rifiles, massValues=c(144, 145, 100, 218), timeRange=c(255, 265),
             pch=1:4, col=1:4)
# display the Retention Index instead
ri_plot_peak(rifiles, massValues=c(144, 145, 100, 218), timeRange=c(255, 265),
             pch=1:4, col=1:4, showRT=FALSE)
# use RI index for the time range
ri_plot_peak(rifiles, massValues=c(144, 145, 100, 218), timeRange=c(267000, 275000),
             useRT=FALSE, pch=1:4, col=1:4)
# use styling per sample
scol <- rep(1:5, each=3)
ri_plot_peak(rifiles, massValues=c(144), timeRange=c(255, 265), pch=19, scol=scol)

# using a tsSample object can achieve the same results (RI files are in text format)
smp <- ImportSamplesFromDir(path, ftype = "text")
ri_plot_peak(smp, massValues=c(144), timeRange=c(255, 265), pch=19, scol=scol)
```

rt2ri *Retention Time to Retention Time Index conversion*

Description

Convert retention times to retention indices based on observed FAME RI and their standard values.

Usage

```
rt2ri(rtTime, observed, standard)
```

Arguments

`rtTime`	The extracted RT's to convert
`observed`	The observed FAME RT's
`standard`	The standard RI for each FAME

Details

Linear interpolation, interpolation outside bounds are done with continued linear interpolation from the last two FAME's

Value

The converted RI

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`RIcorrect`, `FAMEoutliers`

Examples

```
# RI standards
standard <- c(100, 200, 300, 400, 500)
# observed standard retention times
observed <- c(10.4, 19.3, 32.4, 40.2, 50.3)
# a random set of retention times
RT       <- runif(100,1,60)
# the corrected RIs
RI       <- rt2ri(RT, observed, standard)
```

`sampleRI` *Sample specific RI detection*

Description

Return a matrix of the sample specific retention indices (RI) based on the correlating selective masses.

Usage

```
sampleRI(samples, Lib, r_thres = 0.95, columns = NULL,
         method = "dayNorm", minPairObs = 5, showProgressBar = FALSE,
         makeReport = FALSE, pdfFile = "medianLibRep.pdf")
```

Arguments

`samples`	A `tsSample` object created by `ImportSamples` function.
`Lib`	A `tsLib` object created by `ImportLibrary` function with corrected RI values. See `medianRILib`.
`r_thres`	A correlation threshold.
`columns`	Either `NULL`, a `character` vector, or an `integer` vector. In most cases, leave it as `NULL`. This parameter is used to configure the column names or positions of RI text files. See the documentation on the `text2bin` function for further details.
`method`	Normalization method. Options are `"dayNorm"`, a day based median normalization, `"medianNorm"`, normalization using the median of all the intensities of a given mass, and `"none"`, no normalization at all.
`minPairObs`	Minimum number of pair observations. Correlations between two variables are computed using all complete pairs of observations in those variables. If the number of observations is too small, you may get high correlations values just by chance, so this parameters is used to avoid that. Cannot be set lower than 5.
`showProgressBar`	Logical. Should the progress bar be displayed?
`makeReport`	Logical. If `TRUE` will report the RI deviations for every metabolite in the library.
`pdfFile`	The file name where the report will be saved.

Value

A matrix of correlating selective masses RI. Columns represent samples and rows the median RI of the selective masses.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ImportSamples`, `ImportLibrary`, `medianRILib`, `tsLib`, `tsSample`

Examples

```
require(TargetSearchData)
data(TSExample)

# get RI file path
RI.path <- tsd_data_path()
# update RI file path
RIpath(sampleDescription) <- RI.path
# Import Library
refLibrary        <- ImportLibrary(tsd_file_path('library.txt'))

# get the sample RI
corRI <- sampleRI(sampleDescription, refLibrary, r_thres = 0.95)

# same as above, but changing the correlation threshold and the minimum number
# of observations
corRI <- sampleRI(sampleDescription, refLibrary, r_thres = 0.9,
   minPairObs = 10)
```

`TargetSearch-defunct` *Defunct functions in package 'TargetSearch'*

Description

Functions listed here are defunct and no longer available.

Details

These function have been removed from 'TargetSearch' and no longer available. Use the replacement if any.

- `fixRIcorrection` has been replaced by `fixRI`.
- `TargetSearchGUI` has been removed. Please use regular 'TargetSearch' functions. The old source code is archived at `https://github.com/acinostroza/TargetSearchGUI`.

`TargetSearch-deprecated`
Deprecated functions in package 'TargetSearch'

Description

The functions listed here have been deprecated and may be removed as soon as of the next release.

Details

These functions are deprecated. Use the replacement if available.

- `libId` method has been deprecated, or rather renamed. Use the fully equivalent `makeIndex`.

`text2bin` *Convert RI files from text to binary format and vice versa*

Description

This function converts a list of RI files (also known as peak list files) in text or binary format to binary or text format.

Usage

```
text2bin(in.files, out.files=NULL, columns=NULL)

bin2text(in.files, out.files=NULL)
```

Arguments

`in.files`	A character vector of file paths to the input RI files.
`out.files`	A character vector of file paths. If `NULL`, the input file extensions will be changed accordingly ("txt" to "dat" and vice versa).
`columns`	Either a numeric vector with the positions of the columns for `SPECTRUM`, `RETENTION_TIME_INDEX`, and `RETENTION_TIME`; or a character vector with the respective names of those columns. If `NULL`, then default (hard-coded) values will be used. If an integer vector, the column position must start at zero. In addition, the column names can be set by a global option (see details below).

Details

These functions transform a list of RI files from and to binary to text representation. The format of the input files is detected dynamically and an error will be issued on invalid files.

Transforming a binary file to text might be useful if you need to inspect what a RI file looks in the inside (for example, you need to check that the peak detection was correct). On the other hand, a text file to binary is highly recommended as it is faster to parse than a text file.

For text files, the order of the columns is important (see option `columns` above). The first entry is the spectrum list, followed by the retention time index and the retention time. If the column names are other than `SPECTRUM`, `RETENTION_TIME_INDEX`, and `RETENTION_TIME`, use the respective column names or the column names positions starting at zero (first column is zero, second is one, and so on).

Many functions relay on those column names and having to pass them as arguments on each function is tedious, so the global option `TS_RI_columns` can be set at the beginning, for example:

```
# using column names
options(TS_RI_columns=c('spec_column', 'RI_column', 'RT_column'))

# using column indices (zero-based!)
options(TS_RI_columns=c(1, 2, 0))
```

where "spec_column", "RI_column", and "RT_columns" are the names of the spectrum, retention index and retention time columns.

This command is useful if your RI files were generated by another software. However, it is highly recommended to simple convert those custom RI files into `TargetSearch`'s binary format and do not worry about column names.

Value

A character vector of the created files paths or invisible.

File Format

The so-called RI files contain lists of m/z peaks detected for every ion trace measured in the samples. Historically, the file format was a simple tab-delimited text file in the format described below. Note that the column order could differ and additional columns could be present, but they are ignored.

RETENTION_TIME	SPECTRUM	RETENTION_TIME_INDEX
212.46	250:26 256:26 316:27	221029.7
212.51	114:46 162:30 251:27	221081.3
212.56	319:25	221132.9
212.61	95:38 108:30 262:32 266:27 292:25	221184.5

The retention time is usually represented in seconds, while the retention time in arbitrary units, which depends on the retention time correction standard method (in the table above it is in milliseconds, but other units can be used).

The spectrum column is represented by pairs of m/z and raw intensity (peak height), similarly as the representation of a metabolite library (see `ImportLibrary`). Thus each pair correspond to a peak of the respective ion trace.

The disadvantage of using text files is they are slow to parse, so a binary format was created which represents the peak data as binary vectors so they are fast to parse. These files contain the extension `dat`.

Note

Beware that the respective `tsSample` object may need to be updated by using the method `fileFormat`.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`ImportSamples`, `tsSample`, `RIcorrect`

Examples

```
require(TargetSearchData)
# take three example files from package TargetSearchData
in.files <- tsd_rifiles()[1:3]
# out files to current directory
out.files <- sub(".txt", ".dat", basename(in.files))
# convert to binary format
res <- text2bin(in.files, out.files)
stopifnot(res == out.files)

# convert back to text
```

```
res <- bin2text(out.files)
stopifnot(res == basename(in.files))

# Demonstrate how to use the `columns` option
# make dummy RI file with arbitrary column names and save it
tmp <- data.frame(RT=c(101.5,102.5), SPEC=c('12:100 23:100', '114:46 162:30'), RI=c(300, 400) + .75)

# file must be tab-delimited, unquoted strings and no row names
RI_test <- tempfile(fileext=".txt")
write.table(tmp, file=RI_test, sep="\t", quote=FALSE, row.names=FALSE)

# convert this text file to binary format
## wrong! It fails because of invalid columns
# text2bin(RI_test)

# correct! The columns are correct
text2bin(RI_test, columns=c('SPEC', 'RI', 'RT'))

# same example but using integers (not recommended)
text2bin(RI_test, columns=c(1, 2, 0)) # note they start from zero.

# Alternative, set a global option (so it can be used in a session)
opt <- options(TS_RI_columns=c('SPEC', 'RI', 'RT'))
text2bin(RI_test)

# or using integers (again, not recommended)
options(TS_RI_columns=c(1, 2, 0))
text2bin(RI_test)

# unset options
options(opt)
```

`TSExample` *Example GC-MS data for TargetSearch Package*

Description

A `TargetSearch` example GC-MS data. This datasets contains TargetSearch object examples generated from a E.coli salt stress experiment (See package TargetSearchData).

Usage

```
data(TSExample)
```

Format

The data contains the following objects:

sampleDescription a `tsSample` object. The sample description.

refLibrary a `tsLib` object. The reference library.

rimLimits a `tsRim` object. The RI markers definition.

RImatrix a matrix object. The retention time of the RI markers.

corRI a matrix object. The sample RI.

peakData a `tsMSdata` object. The intensities and RIs of all the masses that were searched for.

metabProfile a `tsProfile` object. The metabolite profile.

Details

This dataset contain only the objects. The actual source files are provided by the package Target-SearchData.

See Also

`ImportLibrary`, `ImportSamples`, `ImportFameSettings`,

Examples

```
# this run an example pipeline
require(TargetSearchData)

## The directory with the NetCDF GC-MS files
cdfpath <- tsd_data_path()
cdfpath
list.files(cdfpath)
# get files from TargetSearchData
samp.file <- tsd_file_path("samples.txt")
rim.file  <- tsd_file_path("rimLimits.txt")
lib.file  <- tsd_file_path("library.txt")

# import files from package
sampleDescription <- ImportSamples(samp.file, CDFpath = cdfpath, RIpath = ".")
refLibrary        <- ImportLibrary(lib.file)
rimLimits         <- ImportFameSettings(rim.file, mass = 87)

# update NCDF4
sampleDescription <- ncdf4Convert(sampleDescription, path=".")

# perform RI correction
RImatrix          <- RIcorrect(sampleDescription, rimLimits, massRange = c(85,320),
                     IntThreshold = 25, pp.method = "ppc", Window = 15)
# update median RI
refLibrary        <- medianRILib(sampleDescription, refLibrary)
# get the sample RI
corRI             <- sampleRI(sampleDescription, refLibrary, r_thres = 0.95)
# obtain the peak Intensities of all the masses in the library
peakData          <- peakFind(sampleDescription, refLibrary, corRI)
# make a profile of the metabolite data
metabProfile      <- Profile(sampleDescription, refLibrary, peakData, r_thres = 0.95)

# show the metabolite profile
profileInfo(metabProfile)
# show the matrix intensities
Intensity(metabProfile)
```

tsLib-class — *Class for representing a reference library*

Description

This is a class representation of a reference GC library.

Details

Objects of this class are normally created by the function `ImportLibrary`, which is the recommended method, though they can be created manually as shown in the examples.

Use one of the methods described below to set or get the information of various slots. Most of them are self-evident, but we describe some methods in the following paragraphs.

Some care is needed when using the methods `quantMass<-`, `selMass<-`, `topMass<-`. In order to be consistent, the first m/z value of the slot `topMass` and `selMass` are equal to `quantMass`, and the values of `selMass` are equal to the first couple of values of `topMass`. In other words, the following constrain is applied.

```
quantMass : x[0]
selMass   : x[0], x[1], ..., x[k]
topMass   : x[0], x[1], ..., x[k], x[k+1], ..., x[n]
```

where `1` <= `k` <= `n` and `x[i]` is a m/z value. Thus, using one these methods will change the values of the other slots. In the future, these methods will be deprecated, so it is better to not rely on them. See the last example on how this can lead to unexpected results.

The `c` method requires that the combination of all objects have unique identifiers (names), or in other words, the objects cannot share identifiers. Duplicated identifiers are not allowed and an error will be thrown. Since `TargetSearch` `2.8.0`, different column names of the `libData` slot (see method `libData`) are allowed, as long as the data types are characters, numeric, or logical. More complex type may or not may work.

Use the `libUID` method to set or get the library unique identifiers (also referred as `libID`). This is a short string used for internal object data checking. It is sometimes useful if these identifiers refer to identifiers of external databases, for example, KEGG identifiers. These identifiers are normally supplied via the `libID` column in the function `ImportLibrary`.

Note that the method `libId` is deprecated and may be removed in a future version of `TargetSearch`. Use the equivalent `makeIndex` instead. This method has nothing to do with unique identifiers like `libUID`.

Slots

`Name`: `"character"`, the metabolite or analyte names.

`RI`: `"numeric"`, the expected retention time indices (RI) of the metabolites/analytes.

`medRI`: `"numeric"`, the median RI calculated from the samples.

`RIdev`: `"matrix"`, the RI deviation windows, k = 1,2,3. A three column matrix

`selMass`: `"list"`, every component is a numeric vector containing the selective masses.

`topMass`: `"list"`, every component is a numeric vector containing the top masses.

`quantMass`: `"numeric"`, the mass used for quantification.

`libData`: `"data.frame"`, additional library information.

`spectra`: `"list"`, the metabolite spectra. Each component is a two column matrix: m/z and intensity.

Methods

`[` `signature(x = "tsLib")`: Selects a subset of metabolites from the library.

`$name` `signature(x = "tsLib")`: Access column name of `libData` slot.

`libId` `signature(obj = "tsLib")`: Deprecated. Use `makeIndex` instead.

`length` `signature(x = "tsLib")`: returns the length of the library. i.e., number of metabolites.

`libData` `signature(obj = "tsLib")`: gets/sets the `libData` slot.

`libName` `signature(obj = "tsLib")`: gets the `Name` slot.

`libRI` `signature(obj = "tsLib")`: gets the `RI` slot.

`libUID` `signature(obj = "tsLib")`: gets the library unique identifiers, also know as `libID`.

`libUID<-` `signature(obj = "tsLib")`: sets the library unique identifiers, also know as `libID`.

`medRI` `signature(obj = "tsLib")`: gets the `medRI` slot.

`refLib` `signature(obj = "tsLib")`: Low level method to create a matrix representation of the library.

`RIdev` `signature(obj = "tsLib")`: gets the RI deviations.

`RIdev<-` `signature(obj = "tsLib")`: sets the RI deviations.

`quantMass` `signature(obj = "tsLib")`: gets the quantification mass.

`quantMass<-` `signature(obj = "tsLib")`: sets the quantification mass.

`selMass` `signature(obj = "tsLib")`: gets the selective masses.

`show` `signature(object = "tsLib")`: show method.

`spectra` `signature(obj = "tsLib")`: gets the spectra.

`topMass` `signature(obj = "tsLib")`: gets the top masses.

`c` Combine two or more `tsLib` objects.

Note

Accessing the object slots directly via the `@` operator is **not supported**, in particular, using it to set slot values, because it could result in invalid objects. Always use the accessor methods to retrieve the information.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ImportLibrary`

Examples

```
showClass("tsLib")

# define some metabolite names
libNames   <- c("Metab1", "Metab2", "Metab3")
# the expected retention index
RI         <- c(100,200,300)
# selective masses to search for. A list of vectors.
selMasses  <- list(c(95,204,361), c(87,116,190), c(158,201,219))
# define the retention time windows to look for the given selective masses.
```

```
RIdev      <- matrix(rep(c(10,5,2), length(libNames)), ncol = 3, byrow = TRUE)
# Set the mass spectra. A list object of two-column matrices, or set to
# NULL if the spectra is not available
spectra    <- NULL
# some extra information about the library
libData    <- data.frame(Name = libNames, Lib_RI = RI)
# create a reference library object
refLibrary <- new("tsLib", Name = libNames, RI = RI, medRI = RI, RIdev = RIdev,
            selMass = selMasses, topMass = selMasses, spectra = spectra, libData = libData)

# get the metabolite names
libName(refLibrary)
# set new names
libName(refLibrary) <- c("Metab01", "Metab02", "Metab03")

# get the expected retention times
libRI(refLibrary)
# set the retention time index for metabolite 3 to 310 seconds
libRI(refLibrary)[3] <- 310
# change the selection and top masses of metabolite 3
selMass(refLibrary)[[3]] <- c(158,201,219,220,323)
topMass(refLibrary)[[3]] <- c(158,201,219,220,323)
# change the retention time deviations
RIdev(refLibrary)[3,] <- c(8,4,1)

# get and change the library identifiers
(gcID <- libUID(refLibrary))
libUID(refLibrary) <- c('GC_1', 'GC_2', 'GC_3')

#####################################################################
# objects can be subset and combined with the `[` and `c` operators
lib1 <- refLibrary[1:2]
lib2 <- refLibrary[3]
lib  <- c(lib1, lib2) # this restores the object refLibrary

#####################################################################
# These examples show how changing a quantitative or selective mass
# could lead to unexpected results.

# show quantMasses
quantMass(refLibrary)

# suppose that we want to change the quant mass of metabolite 1 to 96 due
# to a typo in the library. We could do just
quantMass(refLibrary)[1] <- 96

# however, we still see the mass 95 in the selective and top masses.
selMass(refLibrary)[[1]]
topMass(refLibrary)[[1]]

# to remove the mass 95, set the topMass and selMass explicitly, noting that
# the first masses coincides with 96 (the quantMass)
selMass(refLibrary)[[1]] <- c(96, 204, 361)
topMass(refLibrary)[[1]] <- c(96, 204, 361)
```

tsMSdata-class *Class for representing MS data*

Description

This is a class to represent MS data obtained from the sample.

Details

The method `as.list` converts every slot (`RI`, `RT`, and `Intensity`) of a `tsMSdata` object into a matrix. The converted matrices are stored in a list. Each converted matrix has an attribute called 'index' that relates the metabolite index with the respective rows. The components of the resulting list are named as the slots. If the slot `RT` is not defined or empty, then the output list will have only two components. ('RT' and 'Intensity').

These objects are created by the functions `FindPeaks` and `peakFind` and there is really no need to create them manually, so the examples focus on simple data manipulations.

The assignment methods `retIndex<-`, `retTime<-`, and `Intensity<-` are exposed for convenience, though they should not be needed in general. These objects are tightly coupled to the reference library (`tsLib`) and sample (`tsSample`) objects, so a modification could cause unexpected errors. Use under your own risk if you know what you are doing.

Objects from the Class

Objects be created by calling the functions `FindPeaks` or `peakFind`. See their respective documentation.

Slots

`RI`**:** `"list"`, a list containing an RI matrix, one matrix per metabolite

`RT`**:** `"list"`, a list containing an RT matrix, one matrix per metabolite

`Intensity`**:** `"list"`, a list containing a peak intensity matrix, one matrix per metabolite

Methods

Intensity `signature(obj = "tsMSdata")`: gets the peak intensity list.

Intensity<- `signature(obj = "tsMSdata")`: gets the peak intensity list.

retIndex `signature(obj = "tsMSdata")`: gets RT list.

retIndex<- `signature(obj = "tsMSdata")`: sets the RI list.

retTime `signature(obj = "tsMSdata")`: gets the RT list.

retTime<- `signature(obj = "tsMSdata")`: sets the RT list.

show `signature(object = "tsMSdata")`: show function.

as.list `signature(object = "tsMSdata")`: coerce a list object. See details

Note

For reasons explained in the details section, the methods `retIndex<-`, `retTime<-`, and `Intensity<-` are considered **deprecated** and could be removed without notice in the feature.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

FindPeaks, peakFind

Examples

```
showClass("tsMSdata")

# some example manipulations using object peakData
data(TSExample)
peakData

# extract the intensities of the metabolites
int <- Intensity(peakData)
# `int` is a list of matrices where rows are m/z values and columns are samples
head(int)

# extract retention indices (same structure for the intensities)
ri <- retIndex(peakData)
head(ri)

# coerce the whole object into a list of matrices (see details)
data <- as.list(peakData)

# it is possible to use the assignment methods `retIndex<-` and `Intensity<-`
# in these objects, but it is not recommended (only if you know what you are
# doing) for example for small tweaks (not sure what this accomplishes however)
ri[[1]][1,1] <- 234000
retIndex(peakData) <- ri
```

---

tsProfile-class *Class for representing a MS profile*

Description

This is class used to represent a metabolite profile obtained with TargetSearch. This class contains the data you are most likely to use in downstream data analysis.

Details

The metabolite profile information (slot info) is a data.frame that contains information regarding each target metabolite in the reference library (object tsLib). The information can be accessed with the method profileInfo. The profile include the following columns,

- Name The metabolite/analyte name.
- Lib_RI The reference or expected RI (retention index).
- Mass_count The number of correlating masses.
- Non_consecutive_Mass_count Same as mass count, but discounting consecutive masses that correspond with isotopic peaks, which often correlate anyway.

- `Sample_Count_per_Mass` The number of samples in which each correlating mass was found. The values are separated by colons (`;`). The values of the actual masses are in following column.
- `Masses` The correlating masses separated by colons (`;`).
- `RI` The average RI detected in the samples.
- `Score_all_masses` The similarity score calculated using the average intensity of all the masses that were searched for, regardless of whether they are correlating masses.
- `Score_cor_masses` Same as above, but only correlating masses are considered.

Other columns could be present by they are carried over from the metabolite library `tsLib` object.

If the object is created by the function `ProfileCleanUp` instead, additional columns will be created with the prefix `Cor_` followed by a string, such as `Cor_RI`. These columns contain information of "collapsed" metabolites that correlate to mean that they could conflict. The collapsed values are often separated by a vertical bar `|`.

The slots `profRT`, `profRI` and `profInt` contain averaged (median) values for each the matrices in the slots `RT`, `RI`, and `Intensity`. The values are computed only considering the masses that correlate (the other masses are ignored). They represent an average of RT, RI and Intensity per metabolite per sample, given that each selected mass has their own values and often only and average is needed. These matrices are accessed with the methods `profileRT`, `profileRI` and `profileRT`.

Similar to the `tsMSdata` assignment methods, the "setters" methods are exposed for convenience, though they should not be required. The reason is that all the objects are tightly tied and unexpected modifications could result in unintended errors. Use only if you know what you are doing.

Objects from the Class

Objects of this class are created by the functions `Profile` or `ProfileCleanUp`. There is no need to generate objects using `new` methods.

Slots

`info`: `"data.frame"`, the metabolite profile information. See details.

`RI`: `"list"`, a list of RI matrices, one matrix per metabolite, in which rows represent select/top masses and columns correspond to samples.

`RT`: `"list"`, a list of RT matrices, one matrix per metabolite, in which rows represent select/top masses and columns correspond to samples.

`Intensity`: `"list"`, a list of peak-intensity matrices, one matrix per metabolite, and within each matrix, rows represent select/top masses and columns correspond to samples.

`profRI`: `"matrix"`, the profile RI matrix.

`profRT`: `"matrix"`, the profile RT matrix.

`profInt`: `"matrix"`, the profile Intensity matrix.

Extends

This class extends `tsMSdata` directly. Methods that apply to that class apply to `tsProfile`.

Methods

profileInfo `signature(obj = "tsProfile")`: get the profile information.

profileInfo<- `signature(obj = "tsProfile")`: set the profile information.

profileInt `signature(obj = "tsProfile")`: get the profile intensity matrix.

profileInt<- `signature(obj = "tsProfile")`: set the profile intensity matrix.

profileRI `signature(obj = "tsProfile")`: get the profile RI matrix.

profileRI<- `signature(obj = "tsProfile")`: set the profile RI matrix.

profileRT `signature(obj = "tsProfile")`: get the profile RT matrix.

profileRT<- `signature(obj = "tsProfile")`: set the profile RT matrix.

show `signature(object = "tsProfile")`: the show function.

Note

For reasons explained in the details section, the methods `profileRI<-`, `profileRT<-`, `profileInt<-`, and `profileInfo` are considered **deprecated** and could be removed without notice in the feature.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`Profile`, `ProfileCleanUp`, `tsMSdata`

Examples

```
showClass("tsProfile")

# some example manipulations using object peakData
data(TSExample)
metabProfile

# extract the profile information of the metabolite analysis
info <- profileInfo(metabProfile)
# `info` is a data.frame with the information described in details
head(info)

# extract profiled metabolite intensities
mat <- profileInt(metabProfile)
head(mat)

# It is possible to use the assignment methods such as `retIndex<-`,
# `Intensity<-`, `profileInt`, etc., in these objects, but it isn't
# recommended. Use only if if you know what you are doing.

## Not run:
  profileInt(metabProfile) <- updatedMetabMatrix
  profileInfo(metabProfile) <- updatedMetabInfo

## End(Not run)

# in the mock examples above, the objects updated* are updated data.frame or
# matrices with metabolite results.
```

`tsRim-class` *Class for representing retention index markers*

Description

This is a class to represent retention index markers.

Objects from the Class

Objects can be created by the function `ImportFameSettings` or by calls of the form `new("tsRim", limits = [two column matrix with time limits],standard = [a vector with RI standards], mass = [m/z marker])`.

Slots

`limits`: `"matrix"`, two column matrix with lower and upper limits where the standards will be search. One row per standard.

`standard`: `"numeric"`, the marker RI values.

`mass`: `"numeric"`, the m/z marker.

Methods

`[ signature(x = "tsRim")`: Selects a subset of markers.

`rimLimits signature(obj = "tsRim")`: gets the time limits.

`rimLimits<- signature(obj = "tsRim")`: sets the time limits.

`rimMass signature(obj = "tsRim")`: gets the m/z marker.

`rimMass<- signature(obj = "tsRim")`: sets the m/z marker.

`rimStandard signature(obj = "tsRim")`: gets the standards.

`rimStandard<- signature(obj = "tsRim")`: sets the standards.

`length` Returns the number of retention index standards.

`c` Function to combine multiple objects.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ImportFameSettings`

Examples

```
showClass("tsRim")

# create a rimLimit object:
#  - set the lower (first column) and upper (second column) time limites to
#    search for standards.
Lim <- rbind(c(200, 300), c(400,450), c(600,650))
#  - set the retention indices of the standard
```

```
Std <- c(250000, 420000, 630000)
#  - set the mass marker
mass <- 87
#  - create the object
rimLimits <- new("tsRim", limits = Lim, standard = Std, mass = mass)
#  - get the number of standards using `length`
length(rimLimits)

# sometimes you need to change the limits of a particular standard
rimLimits(rimLimits)[2,] <- c(410, 450)

# to change the mass value
rimMass(rimLimits) <- 85

# to select a subset
rim <- rimLimits[1:2]

# remove a marker (number 3 in this case)
rim <- rimLimits[-3]

# objects can be combined using the `c` method
obj <- new("tsRim", limits = cbind(100 * 1:6, 100 * 1:6 + 20),
           standard = 1000 * 1:6, mass = 100) # a random object
# combine objects
obj <- c(rimLimits, obj)
```

---

tsSample-class *Class for representing samples*

Description

This is a class to represent a set of samples.

Objects from the Class

Objects can be created by the function ImportSamples or by calling the object generator function.

`new("tsSample", Names = [sample names], CDFfiles = [list of CDF file names], RIfiles = [list of RI file names], CDFpath = [CDF files path], RIpath = [RI files path], days = [measurement days], data = [additional sample information], ftype = [RI file format])`

Slots

`Names`: `"character"`, the sample names.

`CDFfiles`: `"character"`, the list of CDF file names.

`RIfiles`: `"character"`, the list of RI file names.

`CDFpath`: `"character"`, CDF files path. Deprecated. See Notes.

`RIpath`: `"character"`, RI file path. Deprecated. See Notes.

`days`: `"character"`, measurement days.

`data`: `"data.frame"`, additional sample information.

Methods

`[` `signature(x = "tsSample")`: Selects a subset of samples.

`$name` `signature(x = "tsSample")`: Access column name of `sampleData` slot.

`CDFfiles` `signature(obj = "tsSample")`: list of CDF files.

`RIfiles` `signature(obj = "tsSample")`: list of RI files.

`RIpath` `signature(obj = "tsSample")`: The RI file path.

`CDFpath` `signature(obj = "tsSample")`: The CDF file path.

`length` `signature(x = "tsSample")`: number of samples.

`sampleData` `signature(obj = "tsSample")`: additional sample information.

`sampleDays` `signature(obj = "tsSample")`: measurement days.

`sampleNames` `signature(obj = "tsSample")`: sample names. The names must be unique

`show` `signature(object = "tsSample")`: the show function.

`fileFormat` `signature(obj = "tsSample")`: Sets or gets the RI file format. Options are either "binary" or "text". See note below.

`c` Function to combine multiple objects.

Notes

The method `fileFormat` only changes the internal information of the file type and not the files themselves. To actually change the files, use the functions `bin2text` and `text2bin`.

Note that the slot `Names` (i.e., the sample names/identifiers) must be unique. This allows sample selection by using sample identifiers as well as indices. Also, if columns are selected, the output will be either a vector or a `data.frame` depending on whether one or more columns were selected. More over, it is required that the rownames of the `data` slot are equal to the sample names slots. This is handled internally. If the `Names` are not provided, then the CDF files are used instead (directories and extension are removed). In this case the file names must be unique.

The slots `CDFpath` and `RIpath` are deprecated and not used. However, the methods to set or get the paths will still work. The file paths is stored on the `CDFfiles` and `RIfiles` slots.

The `c` method requires that the combination of all objects have unique identifiers (names), or in other words, the objects cannot share identifiers. Duplicated identifiers are not allowed and an error will be thrown. This method also requires that the column names of the `data` slot (see method `sampleData`) are the same, because the method calls `rbind` internally.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`ImportSamples`

Examples

```
showClass("tsSample")

# get a list of CDF files from a directory
require(TargetSearchData)
CDFpath <- tsd_data_path()
cdffiles <- dir(CDFpath, "cdf", full=TRUE)
```

```
# define the RI file path
RIpath  <- "."

# create a tsSample object with minimal info
sampleDescription <- new("tsSample", CDFfiles = cdffiles, RIpath = RIpath)

## ## ## ## ##

# Alternatively, the CDF path and CDF file names can be given separately
# (this was the old TargetSearch behavior)
cdffiles <- basename(cdffiles)

# create the sample object
sampleDescription <- new("tsSample", CDFfiles = cdffiles, CDFpath = CDFpath, RIpath = RIpath)

## ## ## ## ##

# More parameters could be defined:
# define the RI files and the RI path
RIfiles <- sub("cdf$", "txt", paste("RI_", cdffiles, sep = ""))
RIpath  <- "."

# get the measurement days (the four first numbers of the cdf files, in this
# example)
days <- substring(cdffiles, 1, 4)

# sample names (must be unique)
smp_names <- paste("Sample", 1:length(sampleDescription), sep = "_")

# add some sample info
smp_data <- data.frame(CDF_FILE =cdffiles, GROUP = gl(5,3))

# create the sample object
sampleDescription <- new("tsSample", Names = smp_names, CDFfiles = cdffiles, CDFpath = CDFpath,
    RIpath = RIpath, days = days, RIfiles = RIfiles, data = smp_data)

# change the file paths (relative to the working path)
CDFpath(sampleDescription) <- "my_cdfs"
RIpath(sampleDescription)  <- "my_RIs"

# change sample Names
sampleNames(sampleDescription) <- sprintf("S%03d", 1:length(sampleDescription))

## sample subsetting.
# select samples 1, 3 and 5
(sampleset <- sampleDescription[c(1, 3, 5)])

# or use sample IDs
(sampleset <- sampleDescription[c("S001", "S003", "S005")])

## extract columns
# select column 'GROUP'
(group <- sampleDescription$GROUP)
# or
(group <- sampleDescription[, 'GROUP'])
```

```
## change the measurement days (sets the same day for all samples)
sampleDays(sampleDescription) <- "1"

## Note: the length of `measurement days` variable must be 1 or equal
## to the number of samples, otherwise an error will be thrown

## objects can be combined with the c() function
samp1 <- sampleDescription[1:5]
samp2 <- sampleDescription[6:10]
samp3 <- sampleDescription[11:15]
samp <- c(samp1, samp2, samp3)
```

`tsUpdate,tsSample-method` *Methods for Updating TargetSearch objects* `tsUpdate`

Description

`tsUpdate` is a generic function which can be used to update and old 'TargetSearch' class definition. Currently, this function is only implemented for `tsSample` objects.

Methods

`signature(obj = "tsSample")` Method to update an old `tsSample` object. A change was introduced starting from 'TargetSearch' version 1.42.0.

Author(s)

Alvaro Cuadros-Inostroza

Examples

```
## Not run:
newObject <- tsUpdate(oldObject)

## End(Not run)
```

`updateRI` *Updating Time Index correction*

Description

This function can be used to correct or adjust the detected retention time index (RI) markers or their location to specific retention times. This function adds on `fixRI()` as it also corrects the RI of the CDF files

Usage

```
updateRI(samples, rimLimits, RImatrix = NULL, quiet = TRUE)
```

Arguments

`samples`	A tsSample object created by ImportSamples.
`rimLimits`	A tsRim object. See ImportFameSettings.
`RImatrix`	An optional matrix. It represents a retention time matrix of the detected retention time markers that was obtained after running RIcorrect
`quiet`	Logical. Do not print a list of converted files.

Details

Sometimes the retention time of the RI markers are not detected correctly, either because there was a problem with the standard, or the time limits of the tsRim object were not set correctly, or simply because the markers are not injected with the samples.

In any case, the retention time correction can be fixed by calling this function. This function works almost exactly like `fixRI()`, in fact, it is called internally, and allows correction of RIfiles and CDFfiles at the same time. Check also the documentation of `fixRI()` for extra details.

The parameters are the tsSample and the tsRim object, with optionally a `RImatrix` to force the location of the markers. The parameter `quiet` can be unset to show what samples are corrected.

If only a subset of samples require correction, then they can be chosen by subsetting the object `sample`.

Neededless to say, this function expect that the CDF files exists and are in the TargetSearch format. If this is not the case, then use the function `fixRI()`, as this function deals only with RI files.

Value

The retention index matrix. If `RImatrix` is not `NULL`, then the output is the same matrix.

Note

It is required that all the sample names of `samples` are contained in the colnames of `RImatrix`, but the reverse is not necessary. The number of columns of the output matrix will match the number of samples. Extra columns in `RImatrix` will be ignored and not returned.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`fixRI()`, `RIcorrect()`, `ImportSamples()`, `ImportFameSettings()`

Examples

```
require(TargetSearchData)
# import refLibrary, rimLimits and sampleDescription.
data(TSExample)
CDFpath(sampleDescription) <- tsd_data_path()

# convert a subset of files to netCDF4
smp <- ncdf4Convert(sampleDescription[1:6], path=".")

# make a copy of the RI markers object
fames <- rimLimits
```

```
# mess up the limits of marker 3 (real value is 369 seconds app.)
rimLimits(fames)[3,] <- c(375, 400)

# run RIcorrect (skip CDF-4 conversion)
RImat <- RIcorrect(smp, fames, Window = 15, IntThreshold = 200)

# fix the limits of marker 3
rimLimits(fames)[3,] <- c(360, 380)

# update RI files and CDF files
RImat <- updateRI(smp, fames)

# Pass a RI matrix for manual adjustment
RImat[, 3] <- c(252, 311, 369)
RImat <- updateRI(smp, fames, RImat)

# To select specific samples, simply use sample subsetting
# Note, RImat2 has only one column in this case.
( RImat2 <- updateRI(smp[3], fames, RImat) )
```

Write.Results *Save TargetSearch result objects into files*

Description

This is a convenient function to save the TargetSearch result into tab-delimited text files.

Usage

```
Write.Results(Lib, metabProfile,
              quantMatrix=c('quantmass', 'maxint', 'maxobs', 'none'),
              prefix = NULL,
              selmass = FALSE)
```

Arguments

`Lib`	A `tsLib` object.
`metabProfile`	A `tsProfile` object. The final result of the package. This object is generated by either `Profile` or `ProfileCleanUp`.
`quantMatrix`	Should an intensity matrix using quantification masses be created? This parameter will be passed to `quantMatrix` and indicates whether the quantification mass should be either taken as is from the `Lib` object (as specified by the method `quantMass`), or chosen based on intensity or observations. The file will have the extension '.profile.quantmatrix.txt', unless this is set to none, in which case the file will not be created.
`prefix`	A character string. This is a file name prefix for the output text files. If it is set to `NULL` or `NA`, then `"TargetSearch-YYYY-MM-DD"` is used by default, where Y/M/D represents the year, month and day of today's date.
`selmass`	Logical. This parameter is passed to the function `quantMatrix`. See that function documentation for its meaning.

Details

The function generates the following tab-delimited text files, where “prefix” corresponds with the selected file prefix.

- `prefix.peak.intensity.txt`. Contains information on peak intensity for each target mass in the library per sample.
- `prefix.peak.RI.txt`. Contains information on retention index (RI) for each target mass in the library per sample.
- `prefix.profile.info.txt`. Contains the profile information for each metabolite. See `Profile` and `ProfileCleanUp` for details.
- `prefix.profile.intensities.txt`. Contains information on profile intensities per metabolite. This value is an average of all correlating masses normalized to the mean.
- `prefix.profile.ri.txt`. Contains information on average retention indices of the correlating masses. This value is an average of all correlating masses.
- `prefix.profile.quantmatrix.txt`. The quantification matrix as tab-delimited text file. See `quantMatrix` for details. If the argument `quantMatrix` is equal to “none”, then this file is not created.

The functions uses `write.table` as backend to write the files with parameters `set="\t"` and `quote=FALSE`.

Value

This function does not return anything. It just prints a message with the saved files.

Author(s)

Alvaro Cuadros-Inostroza, Matthew Hannah, Henning Redestig

See Also

`peakFind`, `Profile`, `ProfileCleanUp`, `tsLib`, `tsMSdata`, `tsProfile`, `quantMatrix`

Examples

```
# load precomputed results from `TSExample`
data(TSExample)

# we need the objects metabProfile and refLibrary. This will create files
# with prefix 'TargetSearch-YYYY-MM-DD' corresponding to today's date
Write.Results(refLibrary, metabProfile)

# change the prefix with the parameter `prefix` to 'my_experiment'
Write.Results(refLibrary, metabProfile, prefix='my_experiment')
```

writeLibText *Save a library object in text format*

Description

This function creates tab delimited text file with library information. The created file can be re-imported with the `ImportLibrary` function.

Usage

```
writeLibText(lib, file)
```

Arguments

`lib`	A `tsLib` object. A metabolite library.
`file`	A string naming the output file.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`tsLib`, `ImportLibrary`

Examples

```
require(TargetSearchData)
# get the reference library file
lib.file  <- tsd_file_path("library.txt")

# Import the reference library
refLibrary <- ImportLibrary(lib.file)

# save it to a file
writeLibText(refLibrary, file="libraryCopy.txt")
```

writeMSP *Save spectra in MSP format to be visualized in NIST*

Description

This function creates MSP format file from peak intensities that can be viewed with NIST.

Usage

```
writeMSP(metlib, metprof, file, append = FALSE)
```

Arguments

`metlib`	A `tsLib` object. A metabolite library.
`metprof`	A `tsProfile` object. Usually the output of `Profile` or `ProfileCleanUp` functions.
`file`	A string naming the output file.
`append`	Logical. If `TRUE` the results will be appended to `file`. Otherwise, it will overwrite the contents of `file`.

Details

The generated file can be used to search in other library spectra using the software *NIST Search Software*. This software is available here (only MS Windows is supported) `https://chemdata.nist.gov/dokuwiki/doku.php?id=chemdata:nistlibs`

Note: the output file usually has extension `.msp`, which conflicts with MS Windows Software Patch files. Care is needed to not accidentally double click the file.

Author(s)

Alvaro Cuadros-Inostroza

See Also

`peakFind`, `Profile`, `ProfileCleanUp`, `tsLib`, `tsMSdata`, `tsProfile`

Examples

```
# here we use precomputed objects, in particular the reference library
#(refLibrary) and the metabolite profile (metabProfile).
data(TSExample)

# to call the function, simply use the aforementioned objects as inputs
# the file 'output_file.msp' can be open in NIST Search Software
writeMSP(refLibrary, metabProfile, file="output_file.msp")
```

Index