Package: RmzTabM (via r-universe)

May 21, 2026

Title R API for the mzTab-M Reference Implementation

Version 0.97.13

Description The RmzTabM package provides functions to read and write files in the HUPO-PSI mzTab-M file format. This package represents the reference implementation for mzTab-M import/export for R. Core, low level functions allow to format and parse the individual sections of the mzTab-M format. These functions are designed to be re-used by other packages implementing import/export of results in mzTab-M.

Depends R (>= 4.4)

Encoding UTF-8

License Artistic-2.0

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

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

SystemRequirements pandoc, quarto

Imports data.table, methods

Suggests testthat, pkgdown, BiocStyle, knitr, rmarkdown, quarto, pander

VignetteBuilder quarto

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.3

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

Date/Publication 2026-05-21 13:47:26 UTC

RemoteUrl https://github.com/rformassspectrometry/RmzTabM

RemoteRef HEAD

RemoteSha 7d5340cd1dba6248f3b09baba0ba644eb938fcd5

Contents

`MTD-contact` *Add or Update contact Metadata in an MTD section*

Description

Sets or updates contact-related metadata fields within an MTD (metadata) in a MTD section. When contact metadata already exists, the function can either replace it entirely or append new values to the existing ones.

Usage

```
setMtdContact(
  x = matrix(),
  name = character(),
  affiliation = character(),
  email = character(),
  orcid = character(),
  replace = FALSE
)

getMtdContact(x = matrix())
```

Arguments

x	A MTD section that stores metadata fields. Defaults to `matrix()`. If all values are `NA`, the function returns `x` unchanged.
name	`character` contact's name.
affiliation	`character` contact's affiliation.
email	`character` contact's e-mail address.
orcid	`character` contact's ORCID identifier.
replace	`logical` flag controlling how pre-existing contact metadata is handled: • `FALSE` (default): new values are appended to any existing values. • `TRUE`: existing instrument metadata is discarded and replaced entirely by the supplied arguments.

Value

- For `setMtdContact()`: the input object `x` updated to include the new or merged contact metadata fields. If `x` is empty, the empty `x`.
- For `getMtdContact()`: returns the contact information.

Author(s)

Gabriele Tomè

Examples

```
x <- mtdSkeleton("001", software = "[MS, MS:1001582, xmcs, 4.0.0]")
## Add contact metadata to an existing mzTab object
mtd <- setMtdContact(x, name = "Name Surname",
          affiliation = "PSI-MS",
          email = "name.surname@mail.com", orcid = "0000-0002-1825-0097")

## Replace all existing contact metadata
mtd <- setMtdContact(mtd, name = "Name Surname",
          affiliation = "PSI-MS",
          email = "name.surname@mail.com",
          orcid = "0000-0002-1825-0097",
          replace = TRUE)


getMtdContact(mtd)
```

MTD-CV *Add or Update Controlled Vocabularies (CV) Metadata in an MTD section*

Description

Sets or updates CV-related metadata fields within an MTD (metadata) section. When CV metadata already exists, the function can either replace it entirely or append new values to the existing ones.

Usage

```
setMtdCv(
  x = matrix(),
  label = character(),
  full_name = character(),
  version = character(),
  uri = character(),
  replace = FALSE
)

getMtdCv(x = matrix())
```

Arguments

x	A MTD section that stores metadata fields. Defaults to `matrix()`. If all values are `NA`, the function returns `x` unchanged.
label	`character` describing the labels of the controlled vocabularies/ontologies used in the *mzTab-M* file (e.g. `"MS"` for PSI-MS).
full_name	`character` describing the full names of the controlled vocabularies/ontologies used.
version	`character` describing the version of the controlled vocabularies/ontologies used. (e.g. `"4.1.11"`)
uri	`character` containing the URIs of the controlled vocabularies/ ontologies used in the mzTab file. Note: For OBO ontologies, always use an OBO PURL rather than raw repository links to ensure long-term stability (e.g. `"https://purl.obolibrary.org/obo/ms.o` For other ontology formats, please use the fully qualified PURL pointing to the ontology file.
replace	`logical` flag controlling how pre-existing CV metadata is handled: • `FALSE` (default): new values are appended to any existing values. • `TRUE`: existing instrument metadata is discarded and replaced entirely by the supplied arguments.

Value

- For `setMtdCv()`: the input object `x` updated to include the new or merged CV metadata fields. If `x` is empty, the empty `x`.
- For `getMtdCv()`: returns the CV information.

Author(s)

Gabriele Tomè

Examples

```
x <- mtdSkeleton("001", software = "[MS, MS:1001582, xmcs, 4.0.0]")
## Add CV metadata to an existing mzTab object
mtd <- setMtdCv(x, label = "MS",
          full_name = "PSI-MS controlled vocabulary",
          version = "4.1.11",
          uri = "https://purl.obolibrary.org/obo/ms.obo")

## Replace all existing CV metadata
mtd <- setMtdCv(mtd, label = "MS",
          full_name = "PSI-MS controlled vocabulary",
          version = "4.1.11",
          uri = "https://purl.obolibrary.org/obo/ms.obo",
          replace = TRUE)

## Get CV infrmation
getMtdCv(mtd)
```

MTD-database *Add or Update Database Metadata in an MTD section*

Description

Sets or updates database-related metadata fields within an MTD (metadata) section. When database metadata already exists, the function can either replace it entirely or append new values to the existing ones.

Usage

```
setMtdDatabase(
  x = matrix(),
  name = character(),
  prefix = character(),
  version = character(),
  uri = character(),
  replace = FALSE
)

getMtdDatabase(x = matrix())
```

Arguments

x	A MTD section that stores metadata fields. Defaults to `matrix()`. If all values are `NA`, the function returns `x` unchanged.
name	`character` with the description of databases used. For cases, where a known database has not been used for identification. (e.g., `"[MIRIAM, MIR:00100079, HMDB, ]"`).
prefix	`character` with the prefix used in the "identifier" column of data tables. (e.g., `"hmdb"`).
version	`character` with the database version is mandatory where identification has been performed. This may be a formal version number (`"1.4.1"`), a date of access (`"2016-10-27"`) (ISO-8601 format) or (`"Unknown"`) if there is no suitable version that can be annotated.
uri	`character` with the URI to the database.
replace	`logical` flag controlling how pre-existing database metadata is handled: • `FALSE` (default): new values are appended to any existing values. • `TRUE`: existing instrument metadata is discarded and replaced entirely by the supplied arguments.

Value

- For `setMtdDatabase()`: the input object `x` updated to include the new or merged database metadata fields. If `x` is empty, the empty `x`.
- For `getMtdDatabase()`: get the database metadata.

Author(s)

Gabriele Tomè

Examples

```
x <- mtdSkeleton("001", software = "[MS, MS:1001582, xmcs, 4.0.0]")
## Add database metadata to an existing mzTab object
mtd <- setMtdDatabase(x, name = "[MIRIAM, MIR:00100079, HMDB, ]",
          prefix = "hmdb",
          version = "3.6",
          uri = "http://www.hmdb.ca/")

## Replace all existing database metadata
mtd <- setMtdDatabase(mtd, name = "[MIRIAM, MIR:00100079, HMDB, ]",
          prefix = "hmdb",
          version = "3.6",
          uri = "http://www.hmdb.ca/",
          replace = TRUE)

## Get the database metadata
getMtdDatabase(mtd)
```

MTD-export *Defining and exporting the mzTab-M metadata table*

Description

The metadata section/table of the mzTab-M definition is comprehensive, but also tricky to define. The *RmzTabM* package provides a variety of utility functions that help defining this information. These might be re-used for software package developers to export metabolomics results from their respective software. Importantly, the helper functions listed here only define the core elements for the MTD section, helping with re-arranging and reformatting information available e.g. in `data.frame` format into the respective fields in the MTD section. Additional (optional) fields might need to be added manually depending on availability for an experiment.

See also the specification of the MTD section for details and more information, in particular on the format of the mzTab-M and on mandatory or optional fields.

Generally, MTD data can be categorized into the following parts:

- *Core information*: general information on the experiment. A minimal set can be created using the `mtdSkeleton()` function, which might be further expanded with additional fields. This section allows to describe the general experimental setup. Also, it should contain references to **all** controlled vocabulary (CV) ontologies used and refered to in the mzTab-M file.
- *Sample information*: optional information on individual samples that were measured with the various *assays*/*runs*. The `mtdSample()` function assists in compiling the information for this section.
- *MS run information*: information on the individual MS *runs* (measurements of the samples). Each data file is one run. Use the `mtdMsRun()` function to define this part of the metadata section.
- *Assay information*: the `mtdAssay()` function assists in compiling the assay section of the metadata. Mandatory fields are the name (ID) of the assay and the reference to the *MS run* in which the assay was measured. Optional information on sample reference, external links or custom information can be provided too. In most cases (except multiplexed assays or pre-fractionated samples) one assay will link to one MS run. Each assay **must** represent one column in the following *abundance matrix* sections.
- *Study variable information*: the `mtdStudyVariables()` function allows to format study variable information from an experiment into the mzTab-M format. All study variables need to be assigned to at least one assay and must also be reported in the subsequent abundance matrices.

The helper function listed above can be used sequentially to create the metadata information. See the examples below for a general approach how to define the MTD section of an experiment.

In addition, various helper functions are available to assist in MTD data generation:

- `mtdSort()`: to sort the MTD `matrix` into the expected order.
- `mtdFields()`: helps formatting values into the mzTab-M-specific format.
- `setMtdInstrument()`/`getMtdInstrument()`: set/get Instrument Metadata in an MTD section.
- `setMtdDatabase()`/`getMtdDatabase()`: set/get Database Metadata in an MTD section.

- setMtdCv()/getMtdCv(): set/get Controlled Vocabularies (CV) Metadata in an MTD section.
- setMtdContact()/getMtdContact(): set/get contact Metadata in an MTD section.
- setMtdField()/getMtdField(): set/get a Metadata Field in an MTD section.

Note

The general relationship between *ms_run*, *assay* and *sample*:

- one *ms_run* is the measurement of one assay. In most use cases one *ms_run* is associated with one *assay*.
- one assay can be measured by several MS runs (if fractionated) or multiple assays can be measured in the same MS run (if multiplexed).
- one assay is (generally) one sample, but the same sample can be measured with multiple assays (i.e., technical replicates).

Author(s)

Philippine Louail, Johannes Rainer

See Also

SMF-export and SML-export for creating and formatting the small molecule feature (SMF) and small molecule (SML) sections.

setMtdInstrument(), MTD-database, MTD-CV, MTD-contact and MTD-field.

Examples

```
## Building the mzTab-M metadata information from a `data.frame` with sample
## information of an experiment. Each row in that `data.frame` is one
## measurement of one sample (i.e., represents one *ms_run*). Columns in
## that `data.frame` provide the phenotypic and experimental variables of
## each sample. The example below represents a simple experiment in which
## 3 samples (e.g. cell lines) were measured. Columns *time_point and
## *cell_count* provide the time point (in hours) when samples were
## collected and the number of cells per sample. In addition, one
## sample has the genotype *WT* and two *KO*. Column `"operator"` contains
## the initials of the researcher extracting the samples
exp <- data.frame(
    sample_name = c("S1_T1", "S1_T2", "S2_T1", "S2_T2", "S3_T1", "S3_T2"),
    sample_id = c("S1", "S1", "S2", "S2", "S3", "S3"),
    time_point = c(0, 6, 0, 6, 0, 6),
    cell_count = c(13000, 8700, 10100, 6000, 14000, 4500),
    genotype = c("WT", "WT", "KO", "KO", "KO", "KO"),
    operator = c("BB", "BB", "BB", "BB", "FB", "FB"),
    file_name = c("s1-t1.mzML", "s1-t2.mzML", "s2-t1.mzML", "s2-t2.mzML",
                  "s3-t1.mzML", "s3-t2.mzML")
)
exp
```

```
################################################################################
## Core metadata information

## We first compile the general metadata information. For the present
## example we assume that we performed only preprocessing of the raw MS
## data using *xcms*, thus we don't specify annotation databases used for
## the compound identification/annotation. These could be provided through
## the `database*` parameters. Also, the quantification method and unit(s)
## could be specified using respective parameters of the function.
mtd <- mtdSkeleton(
    id = "EXP_001",
    software = "[MS, MS:1001582], xcms, 4.0.0")
mtd

## We can add additional content to this *skeleton* MTD information with
## any of the `setMtd*` functions.
## We next add a title and description for the experiment.
mtd <- setMtdField(mtd, field = "title",
    value = "Experiment 1 preprocessed data")
mtd <- setMtdField(mtd, field = "description",
    value = "The preprocessed data of experiment 1 samples.")

## We also add information on the MS instrumentation used
mtd <- setMtdInstrument(mtd,
    name = "[MS, MS:1000449, LTQ Orbitrap,]",
    source = "[MS, MS:1000073, ESI,]",
    analyzer = c("analyzer[1]" = "[MS, MS:1000291, linear ion trap,]"),
    detector = "[MS, MS:1000253, electron multiplier,]",
)

## We can also add contact information
mtd <- setMtdContact(mtd, name = c("frodo", "sauron"),
    affiliation = c("fellowship of the ring", "the dark side"),
    email = c("frodo@shire.net", "sauron@mordor.net"),
    orcid = c("0000-0001-2345-6789", "0000-0001-2345-678X"))

mtd

## Other information, such as employed sample processing methods could be
## added in a similar way.


################################################################################
## Sample information

## We next add sample information to the metadata. In addition to the
## specific sample properties that can be defined using the function's
## parameters, arbitrary custom fields can be defined too. Below we add
## information on sample extraction as custom information.
mtd_s <- mtdSample(
    sample = unique(exp$sample_id),
    species = "[NCBITaxon, NCBITaxon:9606, Homo sapiens, ]",
    tissue = "[BTO, BTO:0000759, liver, ]",
```

```
    cell_type = "[CL, CL:0000182, hepatocyte, ]",
    c("[,,Extraction date, 2011-12-21]",
      "[,,Extraction date, 2011-12-22]",
      "[,,Extraction date, 2011-12-23]")
    )
mtd_s

mtd <- rbind(mtd, mtd_s)


################################################################################
## MS run information

## The MS run information should capture information of each individual
## measurement run on an MS instrument. For this, the original data file
## names and location should be provided as well as the format of the
## data files as well as polarity etc.
mtd_msr <- mtdMsRun(
    location = exp$file_name,
    format = "[MS, MS:1000584, mzML file, ]",
    id_format = "[MS, MS:1000530, mzML unique identifier, ]",
    scan_polarity = "positive")

mtd <- rbind(mtd, mtd_msr)


################################################################################
## Assay information

## Each measurement should be associated to (at least) one assay. For our
## simple example, each row in the `data.frame` represents one assay, with
## each assay being measured in one MS run.
a <- mtdAssay(
    assay = exp$sample_name,
    sample_ref = c("sample[1]", "sample[1]", "sample[2]", "sample[2]",
                   "sample[3]", "sample[3]"),
    ms_run_ref = paste0("ms_run[", seq_len(nrow(exp)), "]")
)
a

mtd <- rbind(mtd, a)


################################################################################
## Study variable information

## Study variables can be defined directly from the experiment `data.frame`.
## In our example we use the columns (information on) `"time_point"`,
## `"genotype"`, `"cell_count"` and `"operator"`. Importantly, the row-order
## of the provided `data.frame` has to match the order of the assays (and
## MS runs).
svar <- mtdStudyVariables(
    exp, groups = c("time_point", "genotype", "cell_count", "operator"))
```

```
svar

## The individual columns got encoded to a `"study_variable_group"` with a
## defined (statistical) type and a datatype. Both are inferred
## automatically from the columns of the provided `data.frame`, but could
## also be manually provided through the respective function arguments.

mtd <- rbind(mtd, svar)

## Finally, the `mtdSort()` function can be used to sort the generated
## two-column matrix in the expected order.
mtd <- mtdSort(mtd)

## This metadata information can next be exported manually, or using the
## dedicated export helper functions to an mzTab-M file.
```

`MTD-field` *Add or Update a Metadata Field in an MTD section*

Description

Sets or updates a generic metadata field within an MTD (metadata) section. When the field already exists, the function can either replace it entirely or append new values to the existing ones.

Usage

```
setMtdField(
  x = matrix(),
  field = character(),
  value = character(),
  replace = FALSE
)

getMtdField(x = matrix(), field = character())
```

Arguments

`x`	A MTD section that stores metadata fields. Defaults to `matrix()`. If all values are `NA`, the function returns `x` unchanged.
`field`	`character(1)` name of the metadata field to set or update. Must be a valid MTD field name. (e.g. `"publication"`)
`value`	`character` value(s) to assign to the field. (e.g. `"https://pubs.acs.org/doi/10.1021/acs.analchem.`
`replace`	`logical` flag controlling how pre-existing field metadata is handled. Valid only for indexed fields, unique fields are always replaced. • `FALSE` (default): new values are appended to any existing values. • `TRUE`: existing instrument metadata is discarded and replaced entirely by the supplied arguments.

Value

- For `setMtdField()`: the input object `x` updated to include the new or merged field metadata. If `x` is empty, the empty `x`.
- For `getMtdField()`: `character()` with the requested metadata.

Author(s)

Gabriele Tomè

Examples

```
x <- mtdSkeleton("001", software = "[MS, MS:1001582, xcms, 4.0.0]")
## Add a metadata field to an existing mzTab object
mtd <- setMtdField(x, field = "publication",
          value = "pubmed:21063943|doi:10.1007/978-1-60761-987-1_6")

## Replace all existing values for a metadata field
mtd <- setMtdField(mtd, field = "custom",
          value = "[,,MS operator, Florian]",
          replace = TRUE)

getMtdField(mtd, field = "mzTab-ID")
```

`mtdAssay` *mzTab-M* assay *metadata information*

Description

The `mtdAssay()` function assists in compiling the *assay* information of the metadata section. Each assay **must** be associated with at least one entry of the *ms_run* section (see `mtdMsRun()`). This mapping can be defined with the `ms_run_ref` parameter by providing the ID/name of the run (e.g. `"ms_run[1]"`).

Important: to support the optional additional parameters passed along with `...` **all** parameters (such as `assay`, `sample_ref` etc) have to be **fully** spelled out.

For details and expected input for the various parameter it is **strongly suggested** to consult the mzTab-M documentation.

Usage

```
mtdAssay(
  ...,
  assay = character(),
  external_uri = character(),
  sample_ref = character(),
  ms_run_ref = character(),
  protocol_ref = character(),
  parameters = character()
)
```

Arguments

...	optional additional (custom) parameters for each assay. If provided, the length of the character vector(s) have to match the length of parameter assay.
assay	character with the names of the assay(s). Each assay **must** be reported in the following sections (e.g. the SMF section).
external_uri	optional character with a reference to further information about the assay, for example via a reference to an object within an ISA-TAB file. Can be of length 1 (in which case the same reference is assigned to every assay) or length equal to the length of assay.
sample_ref	optional character with the ID/name of the sample for the assay (e.g. "sample[1]"). If provided, its length has to match the length of assay.
ms_run_ref	character with the ID of associated *ms_run*(s). For multiplexed assays, different assays can refer to the same run. To support pre-fractionated samples, it is also possible to provide a list of character with the runs the assay was measured in. See examples below for more details.
protocol_ref	optional character with the ID/name of the protocol for the assay (e.g. "protocol[1]"). If provided, its length has to match the length of assay.
parameters	optional character with additional parameters of the assay.

Value

two-column character matrix with the content for the assay metadata section.

Author(s)

Johannes Rainer

See Also

MTD-export for other functions defining metadata information

Examples

```
## Minimal example with assay and ms_run_ref defined, each assay assigned
## to its own MS run. Note that for **all** parameters the **full**
## parameter name has to be used (e.g., `assay = `).
mtdAssay(assay = c("a1", "a2", "a3"),
    ms_run_ref = c("ms_run[1]", "ms_run[2]", "ms_run[3]"))

## Example for a multiplexed assay.
mtdAssay(assay = c("a1", "a2", "a3"),
    ms_run_ref = c("ms_run[1]", "ms_run[1]", "ms_run[1]"))

## Example for a pre-fractionated samples
mtdAssay(assay = c("a1", "a2", "a3"),
    ms_run_ref = list(c("ms_run[1]", "ms_run[2]"),
                      c("ms_run[3]", "ms_run[4]"),
                      c("ms_run[5]", "ms_run[6]")))
```

```
## Example adding also sample reference and an external_uri
mtdAssay(
    assay = c("a1", "a2", "a3"),
   external_uri = "https://www.ebi.ac.uk/metabolights/MTBLS517/files/i_Investigation.txt",
    sample_ref = c("sample[1]", "sample[1]", "sample[2]"),
    ms_run_ref = c("ms_run[1]", "ms_run[2]", "ms_run[3]"))

## Example adding also protocol reference
mtdAssay(
   assay = c("a1", "a2", "a3"),
  external_uri = "https://www.ebi.ac.uk/metabolights/MTBLS517/files/i_Investigation.txt",
   sample_ref = c("sample[1]", "sample[1]", "sample[2]"),
   ms_run_ref = c("ms_run[1]", "ms_run[2]", "ms_run[3]"),
   protocol_ref = c("protocol[1]", "protocol[1]", "protocol[1]|protocol[2]"))

## Providing additional, custom information for each assay. These can be
## passed as `character` vectors (same length than `assay`!).
mtdAssay(assay = c("a1", "a2", "a3"),
    ms_run_ref = c("ms_run[1]", "ms_run[2]", "ms_run[3]"),
    c("[MS, , Assay operator, Fred Blogs]",
      "[MS, , Assay operator, Fred Blogs]",
      "[MS, , Assay operator, Frodo]"))
```

`mtdFields` *Prepare and format information for the mzTab-M metadata section*

Description

This function assists in creating and formatting information for the mzTab-M metadata section (MTD). It combines and formats the provided input values for a specific field.

See mzTab-M documentation for more information, examples and expected format.

Usage

```
mtdFields(..., field_prefix = "")
```

Arguments

`...`	`character` vector(s), each having the same length, with the values for the field(s). Parameter `field_prefix` defines the name of the field (e.g., `"cv"`). If names are provided (e.g. `label = "my label"`), these are appended to the `field_prefix` (e.g., for `field_prefix = "cv"`, the returned field's name is combined to `"cv[1]-label"`).
`field_prefix`	`character(1)` defining the prefix to be used (e.g., `prefix = "cv"`). This is the prefix/first part of the field's name.

Value

two column `character matrix` with the formatted elements.

Author(s)

Johannes Rainer, Philippine Louail

See Also

MTD-export for other functions defining metadata information

Examples

```
## Define the CV element with 3 CV terms:
mtdFields(
    label = c("a", "b", "c"),
    full_name = c("A", "B", "C"),
    version = c(1, 2, 3),
    uri = c("u1", "u2", "u3"),
    field_prefix = "cv")

## Define a single software:
mtdFields("[MS, MS:1002879, Progenesis QI, 3.0]", field_prefix = "software")

## Define two softwares:
mtdFields(c("[MS, MS:1002879, Progenesis QI, 3.0]", "[a, b, c, d]"),
    field_prefix = "software")

## Define a software with the optional setting
mtdFields(c("[MS, MS:1002879, Progenesis QI, 3.0]", "[a, b, c, d]"),
    `setting[1]` = c("my cool settings", "none"),
    `setting[2]` = c("other setting", "none"),
    field_prefix = "software")

## Define database fields
mtdFields(
   c("[MITIAM, MRI:00100079, HMDB, ]", "[,, de novo, ]"),
   prefix = c("hmdb", "dn"),
   version = c("3.6", "Unknown"),
   uri = c("http://www.hmdb.ca", "null"),
   field_prefix = "database"
)
```

`mtdFromSampleData` *Create the mzTab-M MTD content from a sample data frame*

Description

`mtdFromSampleData()` compiles the *sample*, *ms_run*, *assay* and *study variable* content of the MTD section from a *sample data* `data.frame`. Each row of this `data.frame` (parameter `x`) is expected to represent on MS run (i.e., acquisition of a sample) of an experimen with columns containing information on the individual MS run and the sample(s).

The columns providing information on the MS run can be specified with parameter `msRunCols`. Mandatory columns are *location* and *scan_polarity*.

The columns providing information on the individual *assays* can be defined with parameter `assayCols`. In most cases one assay represents one MS run.

The columns from which sample information can be retrieved can be defined with `sampleCols`. See notes below for more information.

The columns containing experimental/phenotype information can be defined with parameter `groups`.

Arguments defining columns for MS runs, assays and samples are grouped with parameters `msRunCols.`, `assayCols.` and `sampleCols,`. For each a helper function is available providing defaults and assisting in defining the arguments. See section *Defining columns with information on MS runs, assays and samples*

The MTD information is compiled from the sample data `x` as follows:

- MS runs: each row in `x` is added as one MS run. The column in `x` with the respective file name needs to be defined through parameter `msRunCols.`.
- samples: the unique set of rows based on the column(s) defined with `sampleCols.` are added as samples.
- assay: each unique element in the column defined through `assayCols.` (`assay`) is added as one *assay* referencing to samples and MS runs. In most cases the number of MS runs will match the number of assays, i.e., each row in `x` is one MS run as well as one assay. A 1:n mapping between assay and MS run is also possible.
- study variables: each column defined with parameter `groups` is added as one *study variable group*.

Usage

```
mtdFromSampleData(
  x,
  sampleCols. = sampleCols(),
  msRunCols. = msRunCols(),
  assayCols. = assayCols(),
  groups = character(),
  group_description = character(),
  group_type = character(),
  group_datatype = character(),
  group_unit = character(),
  average_function = "[MS, MS:1002962, mean, ]",
  variation_function = "[MS, MS:1002963, variation coefficient, ]",
  description = character()
)

sampleCols(
  sample = "sample",
  species = "species",
  tissue = "tissue",
  cell_type = "cell_type",
  disease = "disease",
```

```
  description = "description",
  ...
)

msRunCols(
  location = "location",
  instrument_ref = "instrument_ref",
  format = "format",
  id_format = "id_format",
  fragmentation_method = "fragmentation_method",
  scan_polarity = "scan_polarity",
  hash = "hash",
  hash_method = "hash_method"
)

assayCols(
  assay = "assay",
  external_uri = "external_uri",
  sample_ref = "sample_ref",
  ms_run_ref = "ms_run_ref",
  ...
)
```

Arguments

x	data.frame with information on samples, MS runs and assays. Each row is expected to represent one MS run (MS data file) and columns providing information on the measured sample(s) along with experimental and technical information.
sampleCols.	named character vector defining the columns in x containing information for the individual sample fields (with names being the name of the mzTab-M field and values the respective column name in x). The sampleCols() function can be used to define this parameter. See examples below and mtdSample() for more information.
msRunCols.	named character vector defining the columns in x containing information on the MS runs (with names being the name of the mzTab-M field and values the respective column name in x). Required fields/parameters are location and scan_polarity. The msRunCols() function can be used to define this parameter. See examples below and mtdMsRun() for more information.
assayCols.	named character vector defining the columns in x containing information for the individual assays fields (with names being the name of the mzTab-M field and values the respective column name in x). Parameter/field assay is required. The assayCols() function can be used to define this parameter. See examples below and mtdAssay() for more information.
groups	character with the names of the columns in x that should be considered as *study variable groups*. If not defined (the default) a single study variable group "undefined" and single study variable "undefined" will be used.

group_description
: character with an optional description of each study variable group. If provided its length has to match the length of parameter groups.

group_type
: character defining the type for each study variable group. If provided its length has to match the length of parameter groups. Supported values are "[STATO, STATO:0000252, categorical variable,]", "[STATO, STATO:0000228, ordinal variable,]" and "[STATO, STATO:0000251, continuous variable,]" for categorical, ordinal or numerical values, respectively. If not provided (the default) the study variable group type will be inferred from the data type of the respective columns in x.

group_datatype
: optional character defining the data type of the values (i.e., study variables) for the study variable group. If provided, its length has to match the length of paramter groups. Supported values are "xsd:string", "xsd:integer", "xsd:decimal", "xsd:boolean", "xsd:date", "xsd:time", "xsd:dateTime", "xsd:anyURI", and "Parameter" (for *CV Parameters*). Date, time and dateTime values **must** be encoded in ISO 8601 format. If not provided the type is guessed by the data type of the respective column in x.

group_unit
: optional character defining the unit of the group variable (for numeric data types). If provided, its length has to match the length of parameter groups. NA or "" has to be provided for groups for which no unit should be reported. By default (group_unit = character()) no unit is reported for any group.

average_function
: optional character defining the function used to calculate the study variable quantification value (reported in the following table(s)). Can be of length 1 or equal to the number of study variables (to allow defining a different function per variable). Use mtdDefineStudyVariables() to get the complete set of study variables for parameters x and groups. Defaults to the arithmetic mean (average_function = "[MS, MS:1002962, mean,]").

variation_function
: optional character defining the function used to calculate the study variable quantification variation value (reported in the following table(s)). Can be of length 1 or equal to the number of study variables (to allow defining a different function per variable). Use mtdDefineStudyVariables() to get the complete set of study variables for parameters x and groups. Defaults to the coefficient of variation (variation_function = "[MS, MS:1002963, variation coefficient,]").

description
: character of length equal to length(sample) with optional description of each sample.

sample
: character with the labels/names of the individual samples.

species
: list of length equal to length(sample) with each element providing the species (eventually multiple) for each sample. Can also be a character of length(sample) to assing a single species to each sample, or a character(1) of length one to assign the same species to every sample.

tissue
: list with the tissue(s) of each sample. The same format as described for parameter species can be used.

cell_type
: list with the cell type(s) of each sample. The same format as described for parameter species can be used.

disease — list with the disease(s) of each sample. The same format as described for parameter species can be used.

... — named character vectors of length equal to the length of parameter sample with optional *custom* information for each individual sample.

location — character with the location (and file name) of the individual runs. Each element will be one run. This parameter is required, set to "null" if the location of the file(s) is not known.

instrument_ref — (optional) integer() with the index of the instrument the run was measured on.

format — (optional) character defining the format of the external MS data file. If specified, also id_format has be be provided. Can be of length 1 or equal to length(location). For data file(s) in mzML format, format = "[MS, MS:1000584, mzML file,]" can be used.

id_format — (optional) character defining the id format used in the external data file. If specified, also format needs to be defined. Can be of length 1 or equal to length(location). For data file(s) in mzML format, format = "[MS, MS:1000530, mzML unique identifier,]" can be used.

fragmentation_method — (optional) list of character defining the type(s) of fragmentation(s) used in a given ms run. Length must match length of location if provided. If no fragmentation was used for a specific file/run use NULL for that list element (position). As example, if two runs are included, the first does not have any fragmentation and for the second CID and HCD was used define list(NULL, c("[MS, MS:1000133, CID,]", "[MS, MS:1000422, HCD,]")).

scan_polarity — character defining the polarity of a run. Can be either "positive" or "negative". Can be of length 1 or equal to length(location).

hash — (optional) character with the hash value of the corresponding external MS data file. If provided, also hash_method needs to be defined. The length of hash has to match the length of location.

hash_method — (optional) character with the hash method used to generate the value in hash. If provided, also hash needs to be defined. The length of hash_method has to match the length of hash.

assay — character with the names of the assay(s). Each assay **must** be reported in the following sections (e.g. the SMF section).

external_uri — optional character with a reference to further information about the assay, for example via a reference to an object within an ISA-TAB file. Can be of length 1 (in which case the same reference is assigned to every assay) or length equal to the length of assay.

sample_ref — optional character with the ID/name of the sample for the assay (e.g. "sample[1]"). If provided, its length has to match the length of assay.

ms_run_ref — character with the ID of associated *ms_run*(s). For multiplexed assays, different assays can refer to the same run. To support pre-fractionated samples, it is also possible to provide a list of character with the runs the assay was measured in. See examples below for more details.

Value

two column `character` `matrix`.

Defining columns with information on MS runs, assays and samples

- `msRunCols()`: function to create a named `character` defining the columns in `x` containinig information for the individual parameters. To change the default for the column containing the MS data file names (default `location = "location"`) to a column called e.g. `"mzml_file"`: `msRunCols(location = "mzml_file")`. Parameters `location` and `scan_polarity` have to be set to match the column names in `x` with the respective information. See examples for more information.
- `assayCols()`: function to define columns containing information on assays. Parameter `assay` has to be adapted. Through `...` it is also possible to define additional columns with optional information on each assay.
- `sampleCols()`: function to define columns containing sample information. Through `...` additional optional columns with sample information can be provided.

Note

The SML and SMF sections must report one column for each **assay**. Thus, each row in the input sample data `x` is expected to be one assay.

In mzTab-M a *sample* is the source of an biological sample. If in an experiment e.g. multiple blood samples are taken at different time points from the same individual, the individual is considered a single sample. Information on the time points can be provided as study variables using the `groups` parameter.

Author(s)

Johannes Rainer

Examples

```
## Defining an example sample data:
## - file: the mzML file, i.e., the *MS run*.
## - name: the name of the measurement. This is also the name of the sample:
##         QC is the pool of all samples, s1 to s4 the ID of the individual.
## - phenotype: defining the biological replicates, 2 for CVD, 2 for CTR.
## - age: covariate, age of the individuals.
## - injection_index: the order in which samples were measured.
sdata <- data.frame(
    file = c("1.mzML", "2.mzML", "3.mzML", "4.mzML", "5.mzML", "6.mzML"),
    name = c("QC", "s1", "s2", "QC", "s3", "s4"),
    phenotype = c(NA, "CVD", "CTR", NA, "CTR", "CVD"),
    age = c(NA, 35, 32, NA, 43, 32),
    injection_index = c(1, 2, 3, 4, 5, 6))
## Add additional required columns:
sdata$polarity <- "positive"

## Add columns with optional, additional information to the individual
```

```
## samples or assays.
sdata$organism <- "[NCBITaxon, NCBITaxon:9606, Homo sapiens, ]"
sdata$assay_info <- c("run1", "run2", "run3", "run4", "run5", "run6")

## Define the columns in `sdata` that provide information on the individual
## samples.
scols <- sampleCols(sample = "name", species = "organism")

## Define the columns in `sdata` that provide MS run information
mscols <- msRunCols(location = "file", scan_polarity = "polarity")

## Define the columns in `sdata` that provide assay information; we use
## the MS run/file name also for the assay name and add an additional
## column with optional content/information.
acols <- assayCols(assay = "file", assay_info = "assay_info")

## Create the MTD section from the `sdata` `data.frame`. Parameter `groups`
## allows to define the columns in `sdata` that should be encoded as
## *study variable groups*.
m <- mtdFromSampleData(sdata, sampleCols = scols, msRunCols = mscols,
    assayCols = acols, groups = c("phenotype", "age", "injection_index"))

## The sample to assay mapping:
## The repeated injection (assay 1 and 4) of the QC sample are assigned
## to the same sample (1).
getMtdField(m, "assay\\[\\d\\]-sample_ref")

## The study variable 2 represents a value of `"CVD"` for the *phenotype*
## study variable group
getMtdField(m, "study_variable\\[2\\]$")

## and this study variable references the 2nd and 6th assay
getMtdField(m, "study_variable\\[2\\]-assay_refs")


## It is also possible to create a MTD section without samples or
## study variables
m <- mtdFromSampleData(
    sdata,
    msRunCols = c(location = "file", scan_polarity = "polarity"),
    assayCols = c(assay = "assay_info"))
m
```

`mtdMsRun` *msTab-M* ms_run *metadata fields*

Description

The `mtdMsRun()` function allows to define and format the *ms_run* fields of the mzTab-M metadata. The information is build on the actual data file names along with optional additional parameters to characterize the MS run(s).

For details and expected input for the various parameter it is **strongly suggested** to consult the mzTab-M documentation.

Usage

```
mtdMsRun(
  location = character(),
  instrument_ref = integer(),
  format = character(),
  id_format = character(),
  fragmentation_method = vector("list", length(location)),
  scan_polarity = character(),
  hash = character(),
  hash_method = character(),
  parameters = character()
)
```

Arguments

location	character with the location (and file name) of the individual runs. Each element will be one run. This parameter is required, set to "null" if the location of the file(s) is not known.
instrument_ref	(optional) integer() with the index of the instrument the run was measured on.
format	(optional) character defining the format of the external MS data file. If specified, also id_format has be be provided. Can be of length 1 or equal to length(location). For data file(s) in mzML format, format = "[MS, MS:1000584, mzML file,]" can be used.
id_format	(optional) character defining the id format used in the external data file. If specified, also format needs to be defined. Can be of length 1 or equal to length(location). For data file(s) in mzML format, format = "[MS, MS:1000530, mzML unique identifier,]" can be used.
fragmentation_method	(optional) list of character defining the type(s) of fragmentation(s) used in a given ms run. Length must match length of location if provided. If no fragmentation was used for a specific file/run use NULL for that list element (position). As example, if two runs are included, the first does not have any fragmentation and for the second CID and HCD was used define list(NULL, c("[MS, MS:1000133, CID,]", "[MS, MS:1000422, HCD,]")).
scan_polarity	character defining the polarity of a run. Can be either "positive" or "negative". Can be of length 1 or equal to length(location).
hash	(optional) character with the hash value of the corresponding external MS data file. If provided, also hash_method needs to be defined. The length of hash has to match the length of location.
hash_method	(optional) character with the hash method used to generate the value in hash. If provided, also hash needs to be defined. The length of hash_method has to match the length of hash.
parameters	(optional) character with additional parameters of the assays.

Value

two column `character` `matrix` with the *ms_run* metadata fields for a mzTab-M file.

Note

At present only a single polarity per run/file is supported.

Author(s)

Johannes Rainer, Philippine Louail

See Also

MTD-export for other functions defining metadata information

Examples

```
## Build a very basic MTD ms_run section for two data files
fls <- c("file:///path/to/file/a.mzML", "file:///path/to/file/b.mzML")
mtdMsRun(location = fls, scan_polarity = "positive")

## Add also instrument reference information
mtdMsRun(location = fls, scan_polarity = "positive", instrument_ref = 1)

## Finally, add a fragmentation method used for the second file - no
## fragmentation was used for the first file, thus `NULL` is specified.
## Parameter `fragmentation_method` expects a `list` as input to support
## also multiple fragmentation methods per MS run.
mtdMsRun(location = fls, scan_polarity = "positive",
    fragmentation_method = list(NULL, "[MS, MS:1000133, CID, ]"))
```

`mtdProtocol` *mzTab-M* protocol *metadata information*

Description

The `mtdProtocol()` function assists in compiling the *protocol* information of the metadata section. Each protocol is referenced from an *assay* section (see `mtdAssay()`).

For details and expected input for the various parameter it is **strongly suggested** to consult the mzTab-M documentation.

Usage

```
mtdProtocol(
  name = character(),
  type = character(),
  description = character(),
  parameters = character()
)
```

Arguments

name	character with protocol name describing one or more steps of an experimental procedure, such as sample preparation, data acquisition or data processing.
type	character with the protocol type, as defined by the parameter. Can be of length 1 or equal to length(name).
description	optional character with the description of the protocol. Can be of length 1 or equal to length(name).
parameters	optional character with additional parameters of the protocol

Value

two-column character matrix with the content for the protocol metadata section.

Author(s)

Gabriele Tomè

See Also

MTD-export for other functions defining metadata information

Examples

```
## Minimal example with protocol.
mtdProtocol(name = c("protocol1", "protocol2", "protocol3"),
            type = c("[,,,type1]", "[,,,type2]", "[,,,type3]"))

## Example with all the fields
mtdProtocol(name = c("protocol1", "protocol2", "protocol3"),
            type = c("[,,,type]", "[,,,type2]", "[,,,type3]"),
            description = c("description1", "description2", "description3"),
            parameters = list(c("param1.1", "param1.2"), "param2", "param3"))
```

---

mtdSample *msTab-M* sample *metadata information*

Description

The mtdSample() function aids in creating and formatting the (optional) sample information from the mzTab-M metadata section. If defined, the sample information **must** be correctly linked to from the *assay* section. In particular, the assays need to link to the index of the samples defined in this section. One entry for each originating sample should be defined (without information on experimental properties). For each sample one or more additional characteristics (such as species, tissue, cell_type or disease) can be provided. Thus, these parameters expect the input be provided as a list. In addition, if a single value needs to be assigned to each sample, a character(1) of length 1 can be provided with the respective input parameter.

Important: to support the optional additional parameters passed along with ... **all** parameters (such as `sample`, `species` etc) have to be **fully** spelled out.

For details and expected input for the various parameter it is **strongly suggested** to consult the mzTab-M documentation.

Usage

```
mtdSample(
  ...,
  sample = character(),
  species = list(),
  tissue = list(),
  cell_type = list(),
  disease = list(),
  description = character()
)
```

Arguments

`...`	named `character` vectors of length equal to the length of parameter `sample` with optional *custom* information for each individual sample.
`sample`	`character` with the labels/names of the individual samples.
`species`	`list` of length equal to `length(sample)` with each element providing the species (eventually multiple) for each sample. Can also be a `character` of `length(sample)` to assing a single species to each `sample`, or a `character(1)` of length one to assign the same species to every sample.
`tissue`	`list` with the tissue(s) of each sample. The same format as described for parameter `species` can be used.
`cell_type`	`list` with the cell type(s) of each sample. The same format as described for parameter `species` can be used.
`disease`	`list` with the disease(s) of each sample. The same format as described for parameter `species` can be used.
`description`	`character` of length equal to `length(sample)` with optional description of each sample.

Value

two column `character` `matrix` with the information formatted as sample section of the mzTab-M format.

See Also

MTD-export for other functions defining metadata information

Examples

```
## Example sample description data.frame for an experiment
pd <- data.frame(
    sample_name = c("ind_1", "ind_2", "ind_1", "ind_2"),
    sample_id = c("i1_t1", "i2_t2", "i1_t2", "i2_t2"),
    time_point = c(1, 2, 1, 2))

## Define a minimal sample information with just the sample names.
mtdSample(unique(pd$sample_name))

## Add also species information: each sample from the same species
mtdSample(
    sample = unique(pd$sample_name),
    species = "[NCBITaxon, NCBITaxon:9606, Homo sapiens, ]")

## Assume first sample is a mixture of two species
mtdSample(
    sample = unique(pd$sample_name),
    species = list(c("[NCBITaxon, NCBITaxon:9606, Homo sapiens, ]",
                     "[NCBITaxon, NCBITaxon:39767, Human rhinovirus 11, ]"),
                   "[NCBITaxon, NCBITaxon:9606, Homo sapiens, ]")
)

## Add full information including tissue, cell type and disease
mtdSample(
    sample = unique(pd$sample_name),
    species = list(c("[NCBITaxon, NCBITaxon:9606, Homo sapiens, ]",
                     "[NCBITaxon, NCBITaxon:39767, Human rhinovirus 11, ]"),
                   "[NCBITaxon, NCBITaxon:9606, Homo sapiens, ]"),
    tissue = "[BTO, BTO:0000759, liver, ]",
    cell_type = "[CL, CL:0000182, hepatocyte, ]",
    disease = list(c("[DOID, DOID:684, hepatocellular carcinoma, ]",
                     "[DOID, DOID:9451, alcoholic fatty liver, ]"),
                   NULL)
)

## Add also additional custom variables
mtdSample(sample = c("A", "B"),
    c("[,,Extraction date, 2011-12-21]",
      "[,,Extraction date, 2011-12-22]"),
    c("[,,Extraction reason, liver biopsy]",
      "[,,Extraction reason, liver biopsy]"))
```

`mtdSkeleton` *Create a skeleton MTD section with general information*

Description

This *core* MTD section allows to describe the general experimental setup and provides general information of the data set. It should contain references to **all** controlled vocabulary (CV) ontologies used and refered to in the mzTab-M file. The `mtdSkeleton()` function creates a two-column

matrix with the basic mzTab-M *MTD* section based on the provided data. The returned result contains only minimal information. It should be expanded, corrected and completed with additional fields and information (i.e., the *skeleton* returned by this function should be completed with *flesh*).

For details and expected input for the various parameter it is **strongly suggested** to consult the mzTab-M documentation.

Usage

```
mtdSkeleton(
  id = character(),
  software = character(),
 quantification_method = "[MS, MS:1001834, LC-MS label-free quantitation analysis, ]",
  cv_label = c("MS", "PRIDE", "STATO"),
  cv_full_name = c("PSI-MS controlled vocabulary",
    "PRIDE PRoteomics IDEntifications (PRIDE) database controlled vocabulary",
    "General purpose STATistics Ontology"),
  cv_version = c("4.1.138", "16:10:2023 11:38", "2026-04-20"),
 cv_uri = c("https://raw.githubusercontent.com/HUPO-PSI/psi-ms-CV/master/psi-ms.obo",
    "https://www.ebi.ac.uk/ols/ontologies/pride",
    "https://www.ebi.ac.uk/ols4/ontologies/stato"),
  database = c("[,, \"no database\", null ]"),
  database_prefix = c("null"),
  database_version = c("Unknown"),
  database_uri = c("null"),
  small_molecule_quantification_unit =
    "[PRIDE, PRIDE:0000330, Arbitrary quantification unit, ]",
  small_molecule_feature_quantification_unit =
    "[PRIDE, PRIDE:0000330, Arbitrary quantification unit, ]",
  small_molecule_identification_reliability =
    "[MS, MS:1002896, compound identification confidence level, ]",
  mztab_version = "2.1.0-M"
)
```

Arguments

id	character(1) (**mandatory**) with the ID of the data set.
software	character (**mandatory**) with the software(s) used. Can be of length > 1 if multiple softwares were used. Software should be provided in the order in which they were used.
quantification_method	character(1) defining the quantification method used in the experiment.
cv_label	character describing the labels of the controlled vocabularies/ontologies used in the mzTab file as a short-hand, e.g. cv_label = "MS" for PSI-MS.
cv_full_name	character with the full names of the controlled vocabularies/ontologies used in the mzTab file.
cv_version	character with the version of the used vocabularies/ontologies.
cv_uri	character with the URIs of the vocabularies/ontologies.

database character defining the database used for annotation. If no annotation/identification was performed then "[,, no database, null]" should be used.

database_prefix
character defining the prefix used in the *identifier* column of data tables. For *no database*, "null" must be used.

database_version
character with the database version used.

database_uri character with the URI to the database(s). For *no database* "null" must be used.

small_molecule_quantification_unit
character(1) defines the type of units are reported in the small molecule summary quantification/ abundance fields.

small_molecule_feature_quantification_unit
character(1) defines what type of units are reported in the small molecule feature quantification / abundance fields.

small_molecule_identification_reliability
character(1) defines the system used for giving reliability / confidence codes to small molecule identifications MUST be specified if not using the default codes.

mztab_version character(1) defining the mzTab-M version of the file.

Value

two-column character matrix that should be expanded with additional fields (such as *title*, *description* etc) and information (with the help from the mtdFields() function).

Author(s)

Philippine Louail, Johannes Rainer

See Also

MTD-export for other functions defining metadata information

Examples

```
## Define a minimal mzTab-M metadata information
mtd <- mtdSkeleton(id = "001", software = "[MS, MS:1001582, xmcs, 4.0.0]")

## Column 1 has the field names
mtd[, 1]

## Column 2 the respective values
mtd[, 2]

## Add additional fields as defined in the mzTab-M definition
mtd <- rbind(
    mtd,
    c("title", "My simple xcms preprocessed data"),
```

```
    c("description", "A simple example xcms preprocessing."))

tail(mtd)

## Add instrument information
instr <- mtdFields(
    name = "[MS, MS:1000449, LTQ Orbitrap,]",
    source = "[MS, MS:1000073, ESI,]",
    `analyzer[1]` = "[MS, MS:1000291, linear ion trap,]",
    detector = "[MS, MS:1000253, electron multiplier,]",
    field_prefix = "instrument"
)
instr

## Add this information to the metadata
mtd <- rbind(mtd, instr)

## Define sample processing fields using the mtdFields function
sp <- mtdFields(
    c("[MSIO, MSIO:0000146, centrifugation,]",
      "[MSIO, MSIO:0000141, metabolite extraction,]",
      "[MSIO, MSIO:0000141, silylation,]"),
    field_prefix = "sample_processing")
sp

## Add this information to the metadata
mtd <- rbind(mtd, sp)

## Since a new ontology was used for the sample processing, we need also to
## add that to the metadata. We manually define the fields to add using
## `"cv[3]"` because there are already 2 CVs defined in the MTD skeleton.
cv2 <- rbind(
    c("cv[3]-label", "MSIO"),
    c("cv[3]-full_name", "Metabolomics Standards Initiative Ontology"),
    c("cv[3]-version", "1.0.1"),
    c("cv[3]-uri", "http://purl.obolibrary.org/obo/msio.owl")
)

## Add this information to the metadata
mtd <- rbind(mtd, cv2)

## Finally sort the metadata fields according to the expected order
mtd <- mtdSort(mtd)
mtd
```

---

mtdSort *Sort rows in a MTD matrix to match the expected order*

Description

Helper function to sort a mzTab-M *MTD* `matrix`, such as generated by `mtdSkeleton()`, into the correct order of the metadata fields.

Usage

```
mtdSort(x)
```

Arguments

`x`	two-column matrix with the first column containing the metadata field names.

Value

input parameter `x` sorted into the correct order.

Author(s)

Johannes Rainer

See Also

MTD-export for other functions defining metadata information

`mtdStudyVariables` *mzTab-M* study variables *metadata information*

Description

mzTab-M (version >= 2.1) encodes the experimental design of a data set/study using *study variable groups* and *study variables*. The study variable group represents the phenotypic (or experimental) condition and the study variable the actual *value* of a sample (or assay) for that study group.

In R, the most common representation of an experimental design is a `data.frame` where rows are individual samples (assays) and columns the experimental or phenotypic conditions (variables). The `mtdStudyVariables()` takes such a `data.frame` as input and encodes it into the mzTab-M format. Additional parameters such as `group_description`, and `group_type` allow to provide additional information for each study variable group (phenotype) while parameters `average_function`, `variation_function` and `description` can be used to provide properties for the individual study variables. For most experiments the default values of these parameters should suffice.

Usage

```
mtdStudyVariables(
  x,
  groups = character(),
  group_description = character(),
  group_type = character(),
  group_datatype = character(),
  group_unit = character(),
  average_function = "[MS, MS:1002962, mean, ]",
  variation_function = "[MS, MS:1002963, variation coefficient, ]",
  description = character()
)

mtdDefineStudyVariables(x = data.frame(), groups = character())
```

Arguments

x	`data.frame` with rows corresponding to individual *assays* and columns containing the experimental conditions/study variables. The number of rows is thus expected to be the same as the number of assays defined in the *assay* metadata section (using e.g., `mtdAssay()`) and the order of rows is expected to match the order of these.
groups	`character` with the names of the columns in `x` that should be considered as *study variable groups*. If not defined (the default) a single study variable group `"undefined"` and single study variable `"undefined"` will be used.
group_description	`character` with an optional description of each study variable group. If provided its length has to match the length of parameter `groups`.
group_type	`character` defining the type for each study variable group. If provided its length has to match the length of parameter `groups`. Supported values are `"[STATO, STATO:0000252, categorical variable, ]"`, `"[STATO, STATO:0000228, ordinal variable, ]"` and `"[STATO, STATO:0000251, continuous variable, ]"` for categorical, ordinal or numerical values, respectively. If not provided (the default) the study variable group type will be inferred from the data type of the respective columns in `x`.
group_datatype	optional `character` defining the data type of the values (i.e., study variables) for the study variable group. If provided, its length has to match the length of paramter `groups`. Supported values are `"xsd:string"`, `"xsd:integer"`, `"xsd:decimal"`, `"xsd:boolean"`, `"xsd:date"`, `"xsd:time"`, `"xsd:dateTime"`, `"xsd:anyURI"`, and `"Parameter"` (for *CV Parameters*). Date, time and dateTime values **must** be encoded in ISO 8601 format. If not provided the type is guessed by the data type of the respective column in `x`.
group_unit	optional `character` defining the unit of the group variable (for numeric data types). If provided, its length has to match the length of parameter `groups`. `NA` or `""` has to be provided for groups for which no unit should be reported. By default (`group_unit = character()`) no unit is reported for any group.

`average_function`
: optional `character` defining the function used to calculate the study variable quantification value (reported in the following table(s)). Can be of length 1 or equal to the number of study variables (to allow defining a different function per variable). Use `mtdDefineStudyVariables()` to get the complete set of study variables for parameters `x` and `groups`. Defaults to the arithmetic mean (`average_function = "[MS, MS:1002962, mean, ]"`).

`variation_function`
: optional `character` defining the function used to calculate the study variable quantification variation value (reported in the following table(s)). Can be of length 1 or equal to the number of study variables (to allow defining a different function per variable). Use `mtdDefineStudyVariables()` to get the complete set of study variables for parameters `x` and `groups`. Defaults to the coefficient of variation (`variation_function = "[MS, MS:1002963, variation coefficient, ]"`).

`description`
: `character` with a textual description of the study variable. If provided, its length needs to be equal to the number of study variables. Use `mtdDefineStudyVariables()` to get the complete set of study variables for parameters `x` and `groups`. If not provided (the default) the values for the study variable group and study variable are reported.

Details

Each study variable **must** be reported in the abundance tables. Each assay of a data set must be referred to from at least one study variable. Even if a data set has no experimental variables, a study variable group and study variable with the name `"undefined"` **must** be reported. Using `mtdStudyVariables()` without specifying parameter `group` will create such a setup.

The `mtdDefineStudyVariables()` function can be used to get the set (and order) of study variables that would be generated from an input `data.frame` depending on the parameter `groups`.

Value

two-column `character` `matrix` with the content for the study variables metadata section.

Note

Datatypes `"xsd:date"`, `"xsd:time"`, `"xsd:dateTime"` and `"xsd:anyURI"` are currently mapped to `character` in R (and *vice versa*).

At present study variables are mapped to *assays*, but not to *MS runs*.

Author(s)

Philippine Louail, Johannes Rainer

See Also

MTD-export for other functions defining metadata information

Examples

```
## Example phenodata/sample data.frame. Each row is supposed to match
## the measurement of one sample (for a certain condition/time point) from
## one individual
x <- data.frame(
    name = c("I1_0", "I2_0", "I1_6", "I2_6", "I3_0"),
    individual = c("I1", "I2", "I1", "I2", "I3"),
    BMI = c(29.3, 31.4, 29.3, 31.4, 26.5),
    timepoint = c(0, 6, 0, 6, 0),
    T2D = c(TRUE, FALSE, TRUE, FALSE, FALSE)
)

## Study variable groups for this data set could be `"individual"`, `"BMI"`,
## `"timepoint"` and `"T2D"`
mtdStudyVariables(x, groups = c("individual", "BMI", "timepoint", "T2D"))

## Specifying a different average and variation function and selecting
## just two sample columns
mtdStudyVariables(x,
    groups = c("timepoint", "T2D"),
    average_function = "[MS, MS:1002883, median, ]",
    variation_function = "[MS, MS:1002885, standard error, ]")

## Creating a study variable section without defined study variable groups
mtdStudyVariables(x)

## Use `mtdDefineStudyVariables()` to get the definition of study
## variables for a given `x` and `groups`
mtdDefineStudyVariables(x, c("T2D", "BMI", "individual"))
```

`MzTabM` *mzTab-M data container*

Description

The `MzTabM` class is a simple container for the mzTab-M data/file content. Methods for this class allow adding or updating information and validating its content.

New instances can be created using the `MzTabM()` function providing the content for the MTD, SML, SMF and SML sections (through parameters `mtd`, `sml`, `smf`, and `sml`, respectively).

Usage

```
MzTabM(
  mtd = mtdSkeleton(id = "<replace>", software = "<replace>"),
  sml = matrix(ncol = 0, nrow = 0),
  smf = matrix(ncol = 0, nrow = 0),
  sme = matrix(ncol = 0, nrow = 0)
)
```

Arguments

mtd	Two-column matrix or 'data.frame with the MTD content (see MTD-export for details and expected format/content).
sml	matrix or data.frame with the SML content (see SML-export for details and expected format/content).
smf	matrix or data.frame with the SMF content (see SMF-export for details and expected format/content).
sme	matrix or data.frame with the SME content (see SME-export for details and expected format/content).

Adding/getting metadata to/from the MTD section

Various functions are available to get or set metadata information of a MzTabM class:

- getMtdInstrument() and setMtdInstrument() for instrument information.

Author(s)

Johannes Rainer

Examples

```
## Create a minimal mzTab-M with only MTD content.
m <- MzTabM(mtd = mtdSkeleton(id = "001", software = "[,,RmzTabM,]"))
m

## Add instrument information to the MTD section
m <- setMtdInstrument(m, name = "[MS, MS:1000449, LTQ Orbitrap,]",
          source = "[MS, MS:1000073, ESI,]",
          analyzer = c(`analyzer[1]` = "[MS, MS:1000291, linear ion trap,]"),
          detector = "[MS, MS:1000253, electron multiplier,]")
m
getMtdInstrument(m)
```

MzTabM-export *Export a mzTab-M 2.1 file*

Description

Write and validate an mzTab-M (version 2.1) file.

The writer recognises the four standard sections:

- **MTD** (Metadata): Always required. The metadata section provides additional information about the dataset(s) reported in the mzTab-M 2.1 file.
- **SML** (Small Molecule Summary): Optional. Each row of the small molecule section is intended to report one final result to be communicated in terms of a molecule that has been quantified.

- **SMF** (Small Molecule Feature): Optional but required when the SME section is present. The small molecule feature section represents individual MS regions that have been measured/quantified. Each SMF row SHOULD represent a single isotopomer.
- **SME** (Small Molecule Evidence): Optional. The small molecule evidence section represents evidence for identifications of small molecules/features. In a typical case, each row represents one result from a single search or intepretation of a piece of evidence.

Column names of `SMH`, `SFH` and `SEH` are used as section headers.

Usage

```
writeMzTabM(x, path, comments = character())
```

Arguments

`x`	`list` containing the mzTab-M data to be written.
`path`	`character(1)` with the path where save the mzTab-M file.
`comments`	`character` vector of comments to add to the file. The comments will be written after the `MTD` section and before any other section.

Author(s)

Gabriele Tomè

Examples

```
## Basic usage
exp <- data.frame(
    sample_name = c("S1_T1", "S1_T2", "S2_T1", "S2_T2", "S3_T1", "S3_T2"),
    sample_id = c("S1", "S1", "S2", "S2", "S3", "S3"),
    timepoint = c("0h", "6h", "0h", "6h", "0h", "6h"),
    genotype = c("WT", "WT", "KO", "KO", "KO", "KO"),
    operator = c("BB", "BB", "BB", "BB", "FB", "FB"),
    file_name = c("s1-t1.mzML", "s1-t2.mzML", "s2-t1.mzML", "s2-t2.mzML",
                  "s3-t1.mzML", "s3-t2.mzML")
)
mtd <- mtdSkeleton(id = "EXP_1", software = "[MS, MS:1001582, xcms, 4.1.0]")

mtd_msr <- mtdMsRun(
    location = exp$file_name,
    format = "[MS, MS:1000584, mzML file, ]",
    id_format = "[MS, MS:1000530, mzML unique identifier, ]",
    scan_polarity = "positive")
mtd <- rbind(mtd, mtd_msr)

mtd_a <- mtdAssay(
    assay = exp$sample_name,
    sample_ref = c("sample[1]", "sample[1]", "sample[2]", "sample[2]",
                   "sample[3]", "sample[3]"),
    ms_run_ref = paste0("ms_run[", seq_len(nrow(exp)), "]")
)
```

```
mtd <- rbind(mtd, mtd_a)

mtd_svar <- mtdStudyVariables(
    exp, groups = c("timepoint", "genotype", "operator"),
    group_unit = c("[, , hours, ]", "", ""))
mtd <- rbind(mtd, mtd_svar)
mtd <- mtdSort(mtd)

x <- list("MTD" = mtd)
filepath <- file.path(tempdir(), "example.mztab")

result <- writeMzTabM(x, path = filepath)

## Add comments
comments <- c("Test comment")
results <- writeMzTabM(x, path = filepath, comments = comments)
```

`MzTabM-import` *Import a mzTab-M 2.1 file*

Description

Reads and parses an mzTab-M (version 2.1) file into a named list.

The parser recognises the four standard sections:

- **MTD** (Metadata): Always required. The metadata section provides additional information about the dataset(s) reported in the mzTab-M 2.1 file.
- **SML** (Small Molecule Summary): Optional. Each row of the small molecule section is intended to report one final result to be communicated in terms of a molecule that has been quantified.
- **SMF** (Small Molecule Feature): Optional but required when the SME section is present. The small molecule feature section represents individual MS regions that have been measured/quantified. Each SMF row SHOULD represent a single isotopomer.
- **SME** (Small Molecule Evidence): Optional. The small molecule evidence section represents evidence for identifications of small molecules/features. In a typical case, each row represents one result from a single search or intepretation of a piece of evidence.

Section headers (`SMH`, `SFH`, `SEH`) are used as column names for their respective data sections. The `MTD` section does not have a dedicated header row; its two-column key–value structure is returned without column names.

Usage

```
readMzTabM(path, ...)
```

Arguments

path	character(1) with the path to the mzTab-M file.
...	Additional arguments forwarded to every internal call to data.table::fread.

Value

A named list containing between one and four elements, depending on which sections are present in the file:

- MTD: A matrix with two columns. Column names are absent (the prefix column is dropped).
- SML: A matrix whose column names are taken from the SMH header row. Present only when SMH is found in the file.
- SMF: A matrix whose column names are taken from the SFH header row. Present only when SFH is found in the file.
- SME: A matrix whose column names are taken from the SEH header row. Present only when both SFH and SEH are found in the file.

Author(s)

Gabriele Tomè

Examples

```
## Basic usage
result <- readMzTabM(system.file("mztabm/out","xcms_mzTab-M_2-1_v2.mzTab",
                                    package = "RmzTabM"))
names(result)
```

---

parseCvParameter *Utility functions for CV parameters*

Description

mzTab-M makes use of controlled vocabulary (CV) parameters. These parameters are expected to be provided in the format "[CV label, accession, name, value]". In addition, also *user parameters* where only the *name* and *value* are provided are supported, but it is recommendet to use full CV parameters where possible.

parseCvParameter() allows to extract individual fields from a CV parameter.

isCvParameter() tests whether a string is in the expected (CV parameter) format.

Usage

```
parseCvParameter(x, element = 2L)

isCvParameter(x)
```

Arguments

x	character with the CV parameter(s) to parse or test.
element	integer(1) defining which *element* to extract: 1 for the CV label, 2 for the CV term (accession), 3 for the name and 4 for the value element.

Value

character of length(x) with the parsed CV parameter elements or NA_character_ if not present.

Note

While mzTab supports "," in the *name* and *value* field (in which case the respective field must be placed between "), this is currently not supported. Thus, CV label and term (accession) are expected to be correctly extracted, the name and value field might not if "," are present in them.

Author(s)

Johannes Rainer

Examples

```
## Extract CV term
x <- c("[MS, MS:1002962, mean, ]", "[MS, MS:1002883, median, ]")
parseCvParameter(x, 1)

parseCvParameter(x)

parseCvParameter(x, 3)

parseCvParameter(x, 4)

## CV term missing
parseCvParameter("[, , user, value]")
parseCvParameter("[, , user, value]", 3)
parseCvParameter("[, , user, value]", 4)

## Check validity of CV parameters
isCvParameter(c(x, "[a, b, c, d, e]", "d"))
```

---

setMtdInstrument *Add, update or get instrument metadata of an mzTab-M MTD section*

Description

setMtdInstrument() sets or updates instrument-related metadata fields within an MTD (metadata) section. When instrument metadata already exists, the function can either replace it entirely or append new values to the existing ones.

getMtdInstrument() returns the instrument information from an MTD section.

Usage

```
## S4 method for signature 'dfmatrix'
setMtdInstrument(
  x = matrix(),
  name = character(),
  source = character(),
  analyzer = character(),
  detector = character(),
  replace = FALSE
)

getMtdInstrument(x = matrix())
```

Arguments

x	A MTD section that stores metadata fields. Can be a two-column character matrix, a two-column data.frame or a MzTabM() object. Defaults to matrix(). If all values are NA, the function returns x unchanged.
name	character with the name of the instrument used in the experiment. (e.g., "[MS, MS:1000449, LTQ Orbitrap,]").
source	character with the instrument's source used in the experiment. (e.g., "[MS, MS:1000073, ESI,]").
analyzer	character with the instrument's analyzer type(s) used in the experiment. **Must** be provided in the form c("analyzer[1]" = "[MS, MS:1000291, linear ion trap,]") for a single analyzer, or c("analyzer[1]" = "<analyzer 1>", "analyzer[2] = ...") for multiple analyzers.
detector	character with the instrument's detector type used in the experiment. (e.g., "[MS, MS:1000253, electron multiplier,]").
replace	logical flag controlling how pre-existing instrument metadata is handled: • FALSE (default): new values are appended to any existing values. • TRUE: existing instrument metadata is discarded and replaced entirely by the supplied arguments.

Value

- For setMtdInstrument(): the input object x updated to include the new or merged instrument metadata fields. If x is empty, the empty x.
- For getMtdInstrument(): a named character with the instrument information, names being the field names.

Author(s)

Gabriele Tomè

Examples

```
x <- mtdSkeleton("001", software = "[MS, MS:1001582, xmcs, 4.0.0]")
## Add instrument metadata to an existing mzTab object
mtd <- setMtdInstrument(x, name = "[MS, MS:1000449, LTQ Orbitrap,]",
          source = "[MS, MS:1000073, ESI,]",
          analyzer = c(`analyzer[1]` = "[MS, MS:1000291, linear ion trap,]"),
          detector = "[MS, MS:1000253, electron multiplier,]")

## Replace all existing instrument metadata
mtd <- setMtdInstrument(mtd, name = "[MS, MS:1000449, LTQ Orbitrap,]",
          source = "[MS, MS:1000073, ESI,]",
          analyzer = c(`analyzer[1]` = "[MS, MS:1000291, linear ion trap,]"),
          detector = "[MS, MS:1000253, electron multiplier,]",
          replace = TRUE)

x <- mtdSkeleton("001", software = "[MS, MS:1001582, xcms, 4.0.0]")
mtd <- setMtdInstrument(x, name = "[MS, MS:1000449, LTQ Orbitrap,]",
          source = "[MS, MS:1000073, ESI,]",
          analyzer = c(`analyzer[1]` = "[MS, MS:1000291, linear ion trap,]"),
          detector = "[MS, MS:1000253, electron multiplier,]")

getMtdInstrument(mtd)
```

SME-export *Creating the mzTab-M Small Molecule (SME) Table*

Description

The Small Molecule (SME) table contains evidence for annotation (identification) of small molecule features (defined in the SMF section SMF-export). These annotations can result from database searches using MS2 spectra or retention time and *m/z* searches against an in-house annotation database. Each row provides the evidence for one match. Multiple matches/annotations for the same input information can be reported in separate rows (but using the **same** value in the `"evidence_input_id"` column).

The small molecule evidence section **must** always come after the `"Small Molecule Feature"` (SMF) Table. All columns are **mandatory** except for `"opt_"` columns.

A detailed description of the SME format and its columns is provided in the respective section of the mzTab-M specification.

The functions to create and format the SME content are:

- `smeCreate()`: provides a simplified workflow to generate this table in a single step. It takes vectors defining feature properties.

  It automatically:

  - Adds the required `SME_ID` and standard mzTab-M columns (e.g., `evidence_input_id`).
  - Populates missing mandatory columns with `"null"` strings to ensure compliance.
  - Checks content against information available in the MTD section.

  - Sets the line prefix column `SEH` to `"SME"`.
  - Orders columns according to the mzTab-M specification.

  Important: to support the optional additional parameters passed along with `...` **all** parameters have to be **fully** spelled out. All parameters are vectorized and recycled as needed to match the number of rows in the abundance matrix. If their length is not equal to the number of rows or 1, an error is raised.

  See also the specification of the SMF section for details.

- `smeSort()`: can be used to sort the columns of the SME data frame according to the standard order defined in the mzTab-M specification. This is useful if you have added custom columns and want to ensure the standard columns are in the correct order for export.

Usage

```
smeCreate(
  ...,
  evidence_input_id = character(),
  database_identifier = character(),
  chemical_formula = character(),
  smiles = character(),
  inchi = character(),
  chemical_name = character(),
  uri = character(),
  derivatized_form = character(),
  adduct_ions = character(),
  exp_mass_to_charge = numeric(),
  charge = numeric(),
  theoretical_mass_to_charge = numeric(),
  spectra_ref = character(),
  identification_method = character(),
  ms_level = character(),
  id_confidence_measure = NULL,
  rank = 1,
  mtd = NULL
)

smeSort(x)

smeIdConfidenceMeasure(x, mtd, nr = numeric())

smeSpectraRefValidator(x, mtd)
```

Arguments

`...`	optional columns to be added to the SME. The length of arguments passed through `...` has to match `nrow(x)` and the arguments **have** to be named. The name of the argument is used for the column name, prefixed with `"opt_"`.

evidence_input_id
: character with the within-file unique identifier for the input data used to support this identification e.g. fragment spectrum, RT and m/z pair, isotope profile that was used for the identification process. Multiple rows can share the same ID, i.e., if multiple (alternative) annotations would be possible. The identifiers may be human readable but should not be assumed to be interpretable. For example, if fragmentation spectra have been searched then the ID may be the spectrum reference, or for accurate mass search, the "ms_run[2]:458.75".

database_identifier
: character with the putative identification for the small molecule sourced from an external database. This could include additionally a chemical class or an identifier to a spectral library entity, even if its actual identity is unknown. Has to be in the format *:* to provide the database/source of annotation and the ID (e.g. "HMDB:HMDB0001847"). Can be "null" or NA for molecules without annotations. If provided, the length of database_identifier has to match the length of evidence_input_id. If not provided (the default) "null" is assigned to each row/molecule.

chemical_formula
: character with the chemical formula of the compound. This should be specified in Hill notation (EA Hill 1900), i.e. elements in the order C, H and then alphabetically all other elements. Counts of one may be omitted. Elements should be capitalized properly to avoid confusion (e.g., "CO" vs. "Co"). The chemical formula reported should refer to the neutral form. Charge state is reported by the charge field. For example: "N-acetylglucosamine" would be encoded by the string "C8H15NO6". Can be "null" but, if provided, its length has to match the length of evidence_input_id.

smiles
: character with the potential molecule structures in the simplified molecular-input line-entry system (SMILES) for the small molecule. Can be "null" but, if provided, its length has to match the length of evidence_input_id.

inchi
: character with the potential standard IUPAC International Chemical Identifier (InChI) of the given substance. Can be "null" but, if provided, its length has to match the length of evidence_input_id.

chemical_name
: character with the possible chemical/common names for the small molecule, or general description if a chemical name is unavailable. Can be "null" but, if provided, its length has to match the length of evidence_input_id.

uri
: character with the URI pointing to the small molecule's entry in a reference database (e.g., the small molecule's HMDB or KEGG entry). Can be "null" but, if provided, its length has to match the length of evidence_input_id.

derivatized_form
: character with the derivatized form that has been analysed by MS, then the functional group attached to the molecule should be reported using suitable CV terms as appropriate.

adduct_ions
: character() with the assumed classification of molecule's adduct ion after detection, following the general style in the 2013 IUPAC recommendations on terms relating to MS (e.g. "[M+Na]1", "[M+NH4]1+", "[M-H]1-", "[M+Cl]1-"). Can be "null" (or NA) but, if provided, its length has to match the length of evidence_input_id.

exp_mass_to_charge
: numeric with the experimental mass/charge value for the precursor ion. If multiple adduct forms have been combined into a single identification event/search, then a single value e.g. for the protonated form SHOULD be reported here.

charge
: numeric with the small molecule evidence's charge value using positive integers both for positive and negative polarity modes.

theoretical_mass_to_charge
: numeric with the theoretical mass/charge value for the small molecule or the database mass/charge value.

spectra_ref
: character Reference to a spectrum in a spectrum file. If a separate spectrum file has been used for fragmentation spectrum, this MUST be reported in the metadata section as additional ms_runs. The reference must be in the format "ms_run[1-n]:{SPECTRA_REF}". Multiple spectra MUST be referenced using a "|" delimited list for the (rare) cases in which search engines have combined or aggregated multiple spectra in advance of the search to make identifications. If a fragmentation spectrum has not been used, the value should indicate the ms_run to which is identification is mapped e.g. "ms_run[1]".

identification_method
: character with the database search, search engine or process that was used to identify this small molecule (e.g. the name of software, database or manual curation etc). If manual validation has been performed quality, the following CV term SHOULD be used: "quality estimation by manual validation" MS:1001058.

ms_level
: character with the highest MS level used to inform identification (e.g. from an MS2 fragmentation spectrum = "[MS, MS:1000511, ms level, 2]". For direct fragmentation or data independent approaches where fragmentation data is used, appropriate CV terms SHOULD be used.

id_confidence_measure
: matrix with any statistical value or score for the identification. The metadata section reports the type of score used, as "id_confidence_measure[1-n]".

rank
: numeric with the rank of this identification from this approach as increasing integers from 1 (best ranked identification). Ties (equal score) are represented by using the same rank, defaults to "1" if there is no ranking system used.

mtd
: two-column matrix or data.frame with the metadata (MTD) definition of the data set. The first column needs to contain the metadata field names, the second the corresponding values. See MTD-export help page for more information.

x
: for smeSpectraRefValidator(): character with reference to a spectrum in a spectrum file. For smeIdConfidenceMeasure(): matrix or data.frame with any statistical value or score for the identification. For smeSort(): a SML data.frame, such as created by smeCreate().

nr
: numeric with the number of small molecule evidence.

Details

All parameters passed to the smeCreate() function must be **fully named**.

Value

A complete SME `data.frame` ready for export. The data frame contains the 'SEH' line prefix, standard columns ordered according to spec, and any optional columns.

Author(s)

Gabriele Tomè

See Also

MTD-export, SMF-export and SML-export for creating and formatting the metadata (MTD), small molecule feature (SMF) and small molecule (SML) sections.

Examples

```
## Define minimum required MTD section
mtd <- cbind(c("ms_run[1]-location", "ms_run[1]-format",
               "ms_run[1]-id_format", "ms_run[1]-scan_polarity[1]",
               "ms_run[2]-location", "ms_run[2]-format",
               "ms_run[2]-id_format", "ms_run[2]-scan_polarity[1]",
               "ms_run[3]-location", "ms_run[3]-format",
               "ms_run[3]-id_format", "ms_run[3]-scan_polarity[1]"),
             c("1.mzML", "[MS, MS:1000584, mzML file, ]",
               "[MS, MS:1000530, mzML unique identifier, ]",
               "[MS, MS:1000130, positive scan, ]",
               "2.mzML", "[MS, MS:1000584, mzML file, ]",
               "[MS, MS:1000530, mzML unique identifier, ]",
               "[MS, MS:1000130, positive scan, ]",
               "3.mzML", "[MS, MS:1000584, mzML file, ]",
               "[MS, MS:1000530, mzML unique identifier, ]",
               "[MS, MS:1000130, positive scan, ]"))

## Define minimum parameter to build SME section
evidence_input_id = c("ms_run[1]:mass=700.5255;rt=20·5",
                      "ms_run[2]:mass=452.2782;rt=35.1",
                      "ms_run[3]:mass=882.6210;rt=40.0")
exp_mass_to_charge = c(700.5255, 452.2782, 882.6210)
charge = c(1, 1, 1)
theoretical_mass_to_charge = c(700.5281, 452.2777, 882.6224)
spectra_ref = c("ms_run[1]:index=7646", "ms_run[2]:index=7640",
                "ms_run[3]:index=7671|ms_run[3]:index=7725")
identification_method = "[, , LipidDataAnalyzer, 2.11.1]"
ms_level = "[MS, MS:1000511, ms level, 2]"

## Create the final dataframe ready for export
## Note: Fields not provided are automatically set to "null"
sme_final <- smeCreate(
    evidence_input_id = evidence_input_id,
    exp_mass_to_charge = exp_mass_to_charge,
    charge = charge,
    theoretical_mass_to_charge = theoretical_mass_to_charge,
    spectra_ref = spectra_ref,
```

```
    identification_method = identification_method,
    ms_level = ms_level,
    mtd = mtd
)

## The result contains the 'SEH' line prefix and standard columns
head(sme_final)
```

SMF-export *Create the mzTab-M Small Molecule Feature (SMF) Table*

Description

The Small Molecule Feature (SMF) section of the mzTab-M definition captures information on the individual MS features (quantified regions, e.g., elution profiles of specific *m/z* and retention times) that were measured across the assays.

- `smfCreate()` provides a simplified workflow to generate this table in a single step. It takes a matrix of abundances (rows=features, columns=assays) and optional vectors defining feature properties.

  It automatically:

  - Formats the abundance matrix (renaming columns to `abundance_assay[n]`).
  - Adds the required `SMF_ID` and standard mzTab-M columns (e.g., `exp_mass_to_charge`).
  - Populates missing mandatory columns with `"null"` strings to ensure compliance.
  - Sets the line prefix column `SFH` to `"SMF"`.
  - Orders columns according to the mzTab-M specification.

  Important: to support the optional additional parameters passed along with `...` **all** parameters (such as `adduct_ion`, `retention_time_in_seconds` etc) have to be **fully** spelled out. All parameters are vectorized and recycled as needed to match the number of rows in the abundance matrix. If their length is not equal to the number of rows or 1, an error is raised.

  See also the specification of the SMF section for details.

- `smfSort()` can be used to sort the columns of the SMF data frame according to the standard order defined in the mzTab-M specification. This is useful if you have added custom columns and want to ensure the standard columns are in the correct order for export.

Usage

```
smfCreate(
  ...,
  x,
  exp_mass_to_charge = numeric(),
  retention_time_in_seconds = numeric(),
  retention_time_in_seconds_start = numeric(),
  retention_time_in_seconds_end = numeric(),
  SME_ID_REFS = character(),
```

```
  SME_ID_REF_ambiguity_code = character(),
  charge = numeric(),
  adduct_ion = character(),
  isotopomer = character()
)

smfSort(x)
```

Arguments

`...`	Additional optional columns to add. These arguments must be named. The function will automatically prepend `"opt_"` to the names if not already present.
`x`	`matrix` or `data.frame` of abundances. Rows are features, columns are assays. The order of columns is assumed to match the order of assays defined in the Metadata (MTD) section (see `mtdAssay()` for more information).
`exp_mass_to_charge`	`numeric` vector of experimental m/z values. This parameter **must** be provided and can not contain any missing values. Its length has to match `nrow(x)`
`retention_time_in_seconds`	`numeric` vector of retention times in seconds. Defaults to `"null"`.
`retention_time_in_seconds_start`	`numeric` vector of start retention times in seconds (i.e., start retention time of the chromatographic peak of feature). Defaults to `"null"`.
`retention_time_in_seconds_end`	`numeric` vector of end retention times in seconds (i.e., end retention time of the chromatographic peak or feature). Defaults to `"null"`.
`SME_ID_REFS`	`character` vector of SME IDs referencing small molecules. Defaults to `"null"`.
`SME_ID_REF_ambiguity_code`	`character` vector of ambiguity codes for SME ID references. Defaults to `"null"`.
`charge`	`integer` vector of charge states. Defaults to `"null"`.
`adduct_ion`	`character` vector of adducts (e.g. `"[M+H]+"`). Defaults to `"null"`.
`isotopomer`	`character` vector for isotopomer description. Defaults to `"null"`.

Details

All parameters passed to the `smfCreate()` function must be **fully named**.

Value

A complete SMF `data.frame` ready for export. The data frame contains the 'SFH' line prefix, standard columns ordered according to spec, abundance columns, and any optional columns.

Author(s)

Philippine Louail

See Also

MTD-export and SML-export for creating and formatting the metadata (MTD) and small molecule (SML) sections.

Examples

```
## Assume we have a matrix of abundances (e.g., from xcms or similar tools)
## Rows are features, Columns are Samples/Assays.
abund_mat <- matrix(
    c(100.1, 105.2, 110.3,
      200.5, 198.2, 201.0,
      50.0, 55.0, 52.1),
    nrow = 3, byrow = TRUE
)

## Define feature metadata (vectors must match number of rows in matrix)
mz_values <- c(150.05, 200.10, 300.15)
rt_values <- c(20.5, 35.1, 40.0)
adducts <- c("[M+H]+", "[M+Na]+", "[M+H]+")

## Create the final dataframe ready for export
## Note: Fields not provided (like charge) are automatically set to "null"
smf_final <- smfCreate(
    x = abund_mat,
    exp_mass_to_charge = mz_values,
    retention_time_in_seconds = rt_values,
    adduct_ion = adducts,
    ## Optional custom column example
    global_custom_attribute = c("A", "B", "C")
)

## The result contains the 'SFH' line prefix, standard columns, and
## abundances
head(smf_final)
```

`SML-export` *Creating the mzTab-M Small Molecule (SML) Table*

Description

The Small Molecule (SML) table is mandatory in the mzTab-M format and provides abundances along with annotations on the small molecules of an analysis result. All columns in this table need to be tab-delimited and no empty cells are allowed (`"null"` must be used for missing values, for columns that support that).

Each row of the small molecule section is intended to report one final result to be communicated in terms of a molecule that has been quantified. In many cases, this may be the molecule of biological interest. In general, different adduct forms for the same molecule would be reported in the *Small Molecule Feature* (SMF) section.

A detailed description of the SML format and its columns is provided in the respective section of the mzTab-M specification.

The functions to create and format the SML content are:

- `smlCreate()`: main function that creates the SML table based on the provided abundance matrix and small molecule annotation information.

  The function automatically:

  - Formats the abundance matrix `x` (renaming column names to `"abundance_assay[n]"`)
  - Adds the required `"SML_ID"` column with a running integer representing the primary key (unique ID) for each molecule.
  - Populates missing mandatory columns with `"null"` strings.
  - Sets the line prefix column.
  - Orders the columns according to the mzTab-M specification.

  After `smlCreate()`, required study variable abundance columns should be either added manually or using the `smlAddStudyVariableColumns()` function. These required columns contain the average and variation of molecule abundances across samples for a particular study variable.

- `smlAddStudyVariableColumns()`: *completes* the SML table created with `smlCreate()` by adding columns with aggregated assay values (molecule abundances) per study variable(s). Study variables have to be listed in the MTD section provided with the `mtd` parameter (e.g. `"study_variable[1]"`) where also the aggregation functions must be defined (e.g. `"study_variable[1]-average_fu`
  This aggregation and variation functions has to be defined using the respective MS CV ontology term (e.g. `"MS:1002962"` for the mean and `"MS:1002963"` for the coefficient of variation). See MTD-export for details on the MTD section.

  The function:

  - gets all study variables from the metadata section (`mtd`)
    - for each study variable:
    - gets the assay references from MTD
    - gets the summary functions (for average and variation) from MTD (resolving the MS CV term to the respective R function)
    - gets the referred abundance columns in `x`
    - calculates the summary statistic and these as additional columns to `x`.

- `smlSort()`: sorts the columns of an SML table into the expected order.

Usage

```
smlCreate(
  ...,
  x,
  SMF_ID_REFS = character(),
  database_identifier = character(),
  chemical_formula = character(),
  smiles = character(),
  inchi = character(),
  chemical_name = character(),
  uri = character(),
```

```
  theoretical_neutral_mass = numeric(),
  adduct_ions = character(),
  reliability = character(),
  best_id_confidence_measure = character(),
  best_id_confidence_value = character()
)

smlSort(x)

smlAddStudyVariableColumns(x, mtd)
```

Arguments

`...`	optional columns to be added to the SML. The length of arguments passed through `...` has to match `nrow(x)` and the arguments **have** to be named. The name of the argument is used for the column name, prefixed with `"opt_"`.
`x`	for `smlCreate()`: `matrix` or `data.frame` with the abundances of the small molecules. Only the abundances of the molecules should be reported (rows being molecules, columns samples). For `smlAddStudyVariableColumns()`: `data.frame` generated by `smlCreate()`. For `smlSort()`: a SML `data.frame`, such as created by `smlCreate()`.
`SMF_ID_REFS`	`character` with the index (ID) of all features (rows in the SMF matrix) each small molecule bases on. The length of the input has to match `nrow(x)`. If not provided, `"null"` is assigned to all molecules. Multiple IDs for one molecule can be separated by `"|"`.
`database_identifier`	`character` with the identifier of the molecule. Has to be in the format *:* to provide the database/source of annotation and the ID (e.g. `"HMDB:HMDB0001847"`). Multiple values can be concatenated using `"|"`. Can be `"null"` or `NA` for molecules without annotations. The length of `database_identifier` has to match the number of rows of `x`. If not provided (the default) `"null"` is assigned to each row/molecule.
`chemical_formula`	`character` with the (possible) chemical formula(s) of the compound. Can be `"null"` but, if provided, its length has to match the number of rows of `x`. Multiple formulas have to be concatenated with `"|"` and the number of formulas per molecule **have** to match the number of reported identifiers in `database_identifier`.
`smiles`	`character` with the potential molecule structures in the simplified molecular-input line-entry system (SMILES) for the small molecule. Can be `"null"` but, if provided, its length has to match the number of rows of `x`. The number of smiles per molecule **must** match the number of reported identifiers in `database_identifier` and have to be separated by `"|"`.
`inchi`	`character` with the potential standard IUPAC International Chemical Identifier (InChI) of the given substance. Can be `"null"` but, if provided, its length has to match the number of rows of `x`. The number of inchis per molecule **must** match the number of reported identifiers in `database_identifier` and have to be separated by `"|"`.

`chemical_name` `character` with the possible chemical/common names for the small molecule, or general description if a chemical name is unavailable. Can be `"null"` but, if provided, its length has to match the number of rows of `x`. The number of names provided **must** match the number of entities reported under `database_identifier` and have to be separated by `"|"`.

`uri` `character` with the URI pointing to the small molecule's entry in a reference database (e.g., the small molecule's HMDB or KEGG entry). Can be `"null"` but, if provided, its length has to match the number of rows of `x`. The number of uris per molecule **must** match the number of reported identifiers in `database_identifier` and have to be separated by `"|"`.

`theoretical_neutral_mass` `numeric()` with the small molecule's precursor's theoretical **neutral** mass. Can be `NA` but, if provided, its length has to match the number of rows of `x`. The number of masses per molecule **must** match the number of reported identifiers in `database_identifier` and have to be separated by `"|"`.

`adduct_ions` `character()` with a `"|"` separated list of detected adducts for this this molecule. Can be `"null"` (or `NA`) but, if provided, its length has to match the number of rows of `x`.

`reliability` `character()` or `integer()` of length equal to `nrow(x)` with the reliability of annotation following one of the supported schemas. Has to be of length equal to `nrow(x)`. `"null"` or `NA` are allowed if no reliability information is available.

`best_id_confidence_measure` `character()` defining the approach or database search that identified the molecule with highest confidence. Has to be a CV parameter, but can also be `"null"` or `NA` if not available. If provided, the length has to match `nrow(x)`.

`best_id_confidence_value` `numeric()` the value/score of the best confidence measure. If provided, the length has to match `nrow(x)`. The type of score **must** be defined in the metadata section. Can also be `NA` if not available.

`mtd` two-column `matrix` or `data.frame` with the metadata (MTD) definition of the data set. The first column needs to contain the metadata field names, the second the corresponding values. See MTD-export help page for more information.

Details

All parameters passed to the `smlCreate()` function must be **fully named**.

Value

For `smlCreate()`: a `character` `data.frame` ready for export. The `data.frame` contains the `"SMH"` line prefix, the standard columns ordered according to the mzTab-M specification, the abundance columns, the abundance columns aggregated per study variables and any optional columns.

For `smlSort()`: the input `data.frame` with columns ordered as defined by the mzTab-M definition.

Author(s)

Johannes Rainer

See Also

MTD-export and SMF-export for creating and formatting the metadata (MTD) and small molecule feature (SMF) sections.

Examples

```
## Define an example abundance matrix
amat <- matrix(abs(rnorm(20)), ncol = 4, nrow = 5)
colnames(amat) <- c("sample_1", "sample_2", "sample_3", "sample_4")

## Define annotation information for each row (molecule). This information
## should be the result of an annotation pipeline used.
anns <- data.frame(
    hmdb_id = c("HMDB:HMDB0001847",
                NA,
                "HMDB:HMDB0000122|HMDB:HMDB0000169",
                "HMDB:HMDB0000258",
                "HMDB:HMDB0060475"),
    formula = c("C8H10N4O2",
                NA,
                "C6H12O6|C6H12O6",
                "C12H22O11",
                "C5H9NO4"),
    smiles = c("CN1C=NC2=C1C(=O)N(C)C(=O)N2C",
               NA,
          "OC[C@H]1O[C@H](O)[C@H](O)[C@H](O)[C@H]1O|OC[C@H]1O[C@H](O)[C@H](O)[C@H](O)[C@H]1O",
               NA,
               "NC(CCC(O)=O)C(O)=O"),
    neutral_mass = c(194.0804, NA, "180.0634|180.0634", 342.1162, 147.0531),
    name = c("caffeine", NA, "glucose|mannose", "sucrose", "DL-Glutamate"),
    adduct_ions = c("[M+H]1+", NA, "[M+Na]1+",
                    "[M+H]1+", "[M+H]1+"),
    uri = c("http://www.hmdb.ca/metabolites/HMDB0001847",
            NA,
        "http://www.hmdb.ca/metabolites/HMDB0000122|http://www.hmdb.ca/metabolites/HMDB0000169",
            "http://www.hmdb.ca/metabolites/HMDB0000258",
            "http://www.hmdb.ca/metabolites/HMDB0060475"))

## Create the SML table based on this information. Note that we have to
## use the **full** name for **all** function arguments (i.e., `x = amat`
## etc).
sml <- smlCreate(
    x = amat, database_identifier = anns$hmdb_id,
    chemical_formula = anns$formula, smiles = anns$smiles,
    theoretical_neutral_mass = anns$neutral_mass, uri = anns$uri,
    chemical_name = anns$name, adduct_ions = anns$adduct_ions)
sml

## Update the reference between rows in SML with those in an SMF table
sml$SMF_ID_REFS <- c("3", "5", "8|23", "12", "16")

## Re-order columns in the expected order
```

```
sml <- smlSort(sml)
sml

## Add study variable abundance columns based on study variable definition
## in a metadata (MTD) section.
## In the example below we first define a metadata section with the
## minimally required information for `smlAddStudyVariableColumns()`:
## the study variables, the samples (assays) that belong to each and
## a average and variation function (see ?MTD-export for full help on the
## metadata section).
## In the example we assign sample 1 and 3 to the `"male"` category/study
## variable and 2 and 4 to `"female"`. For the average and variation
## function we use the mean and coefficient of variation.
mtd <- cbind(
    c("study_variable[1]",
      "study_variable[1]-assay_refs",
      "study_variable[1]-average_function",
      "study_variable[1]-variation_function",
      "study_variable[1]-description",
      "study_variable[2]",
      "study_variable[2]-assay_refs",
      "study_variable[2]-average_function",
      "study_variable[2]-variation_function",
      "study_variable[2]-description"),
    c("male",
      "assay[1]|assay[3]",
      "[MS, MS:1002962, mean, ]",
      "[MS, MS:1002963, variation coefficient, ]",
      "sex of participants, male",
      "female",
      "assay[2]|assay[4]",
      "[MS, MS:1002962, mean, ]",
      "[MS, MS:1002963, variation coefficient, ]",
      "sex of participants, female")
)
## Add study variable average and variation columns and sort the columns
sml <- smlAddStudyVariableColumns(sml, mtd) |> smlSort()
sml

## The average and variation for each study variable were calculated from
## the assay abundances and added as additional columns to the SML.
```

Index