Package 'CardinalIO'

September 18, 2024

Type Package

Title Read and write mass spectrometry imaging files

Version 1.3.4

Date 2023-8-29

Description Fast and efficient reading and writing of mass spectrometry imaging data files. Supports imzML and Analyze 7.5 formats. Provides ontologies for mass spectrometry imaging.

License Artistic-2.0

Depends R (>= 4.3), matter, ontologyIndex

Imports methods, S4Vectors, stats, utils, tools

Suggests BiocStyle, testthat, knitr, rmarkdown

VignetteBuilder knitr

biocViews Software, Infrastructure, DataImport, MassSpectrometry, ImagingMassSpectrometry

URL http://www.cardinalmsi.org

BugReports https://github.com/kuwisdelu/CardinalIO/issues

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

git_branch devel

git_last_commit af4b497

git_last_commit_date 2024-09-14

Repository Bioconductor 3.20

Date/Publication 2024-09-18

Author Kylie Ariel Bemis [aut, cre]

Maintainer Kylie Ariel Bemis <k.bemis@northeastern.edu>

Contents

`CardinalIO-package` *CardinalIO package*

Description

Read and write mass spectrometry imaging files

Details

`CardinalIO` provides fast and efficient reading and writing of mass spectrometry imaging data files. It supports imzML and Analyze 7.5 formats, and provides ontologies for mass spectrometry imaging.

See `vignette("CardinalIO-guide")` for an introduction to the standard imzML format and how to use `parseImzML` and `writeImzML` to parse and write imzML files.

For a complete list of functions, use `library(help = "CardinalIO")`.

Author(s)

Kylie A. Bemis

`exampleImzMLFile` *Example imzML files*

Description

Get a local file path to an example imzML file originally downloaded from https://ms-imaging.org/imzml/example-files-test/.

Usage

```
exampleImzMLFile(type = c("continuous", "processed"))
```

Arguments

`type`	The type of example imzML file path to return.

Value

A string giving the local file path.

Author(s)

Kylie A. Bemis

See Also

`parseImzML`

Examples

```
# get the path to an example imzML file
path <- exampleImzMLFile("processed")

# parse the file
p <- parseImzML(path)
print(p)
```

---

get_obo *Mass spectrometry imaging ontology*

Description

These functions provide ways of getting and querying the ontologies necessary for imzML. Specifically, ontologies for mass spectrometry imaging ('ims'), mass spectrometry ('ms'), and units ('uo') are provided.

Usage

```
get_obo(obo = c("ims", "ms", "uo"), ...)

valid_terms(terms, obo = c("ims", "ms", "uo"),
    check = c("any", "name", "accession"))

find_terms(pattern, obo = c("ims", "ms", "uo"),
    value = c("name", "accession"))

find_term(term, obo = c("ims", "ms", "uo"),
    value = c("name", "accession"))

find_descendants_in(list, terms, obo = c("ims", "ms", "uo"))
```

Arguments

obo	The ontology to get or use.
terms	One or more ontology terms (either names or accessions) to check for validity in the ontology.
pattern	The regular expression pattern to search in the ontology.
term	An ontology term to partially match (by name, *not* accession).
...	Additional arguments passed to get_ontology) when first loading the ontology.
check	When validating terms, are they names ('name'), accession IDs ('accession') or either ('any')?
value	Should the term names ('name') or accession IDs ('accession') be returned?
list	A named list where the names are accession IDs.

Details

`get_obo()` caches and returns the requested ontology.

`find_term()` and `find_terms()` both query the specified ontology for the given term and return it if found. The former uses partial matching via `pmatch` and must unambiguously resolve to a single term. The latter uses `grep` and finds all potential matching terms.

`find_descendants_in()` finds descendants of particular terms in a named list where the names are accession IDs. It returns the list subsetted to matching descendants.

Value

For `get_obo()`, a `ontology_index` object.

For `valid_terms()`, a logical vector indicating whether the corresponding terms are valid.

For `find_descendants_in()`, a subset of the original list.

For all others, a character vector of the requested terms.

Author(s)

Kylie A. Bemis

See Also

`get_ontology`

Examples

```
# find position-related terms in imaging ontology
find_terms("position", "ims")

# find a specific term's accession ID
find_term("position x", "ims", value="accession")

# find all terms related to a vendor in MS ontology
find_terms("Bruker", "ms")
find_terms("Thermo", "ms")
```

ImzMeta-class *Mass spectrometry imaging experimental metadata*

Description

The `ImzMeta` class provides a simpler and more limited interface for tracking mass spectrometry (MS) imaging experimental metadata compared to a full `ImzML` instance as returned by `parseImzML`. It is a simple list of expected/required metadata tags that can be easily set by the user. Replacement methods support partial matching to identify the correct controlled-vocabulary parameter.

Usage

```
## Instance creation
ImzMeta(...)
```

Arguments

...	Named metadata tags (in the form `name=value`, e.g., `spectrumRepresentation="profile specrum"`. Use `names(ImzMeta())` to see possible tags.

Details

The `ImzMeta` class supports lossy conversion between itself and `ImzML` instances. Only the supported information is captured, so converting from `ImzML` and then back to `ImzML` will lose some information. It is primarily intended for ease of use when preparing the metadata from scratch and when a complete `ImzML` instance is not available at the time of writing the file.

Value

An object of class `ImzMeta`.

Methods

Standard generic methods:

`x$name, x$name <- value`**:** Get or set a tag.

`x[["name"]], x[["name"]] <- value`**:** Get or set a tag.

Note

This class does *not* currently meet minimum reporting guidelines for MS imaging experiments, as that is not its purpose. It is designed to provide the minimum required experimental metadata for writing a valid imzML file. For example, it does not currently support sample metadata, as this would require ontologies that are outside of the scope of the present package. This may be expanded in the future if the need arises.

Author(s)

Kylie A. Bemis

See Also

`parseImzML`, `writeImzML`

Examples

```
## create an empty ImzMeta instance
e <- ImzMeta()

## set some experimental metadata
e$spectrumType <- "MS1 spectrum"
e$spectrumRepresentation <- "profile spectrum"
e

# convert to ImzML instance
as(e, "ImzML")

# convert from a parsed imzML file
path <- exampleImzMLFile()
p <- parseImzML(path)
as(p, "ImzMeta")
```

`parseAnalyze` *Parse an Analyze 7.5 File*

Description

Analyze 7.5 is a format originally designed for magnetic resonance imaging (MRI), but is also used for mass spectrometry (MS) imaging.

Usage

```
parseAnalyze(file, ...)
```

Arguments

`file`	The file path to either of the ".hdr" or ".img" files.
`...`	Not currently used.

Details

Because the Analyze 7.5 is originally intended for MRI, it stores the complete data cube as an N-dimensional array. For MRI data, there are typically 4 dimensions. For MS imaging data, there are typically 3 dimensions, where the first dimension is the m/z value axis, and the other two dimensions are spatial. If a ".t2m" file is present (storing the m/z-values for MS imaging data), then it will be parsed as well.

Value

An object of class `Analyze75`, which is a list with components named `hdr`, `img`, and (if appropriate) `t2m`.

Author(s)

Kylie A. Bemis

Examples

```
# create a toy data cube
set.seed(2023)
nx <- 3
ny <- 3
nmz <- 500
mz <- seq(500, 510, length.out=nmz)
intensity <- replicate(nx * ny, rlnorm(nmz))
dim(intensity) <- c(nmz, nx, ny)
path <- tempfile(fileext=".hdr")

# write it in Analyze 7.5 format
writeAnalyze(intensity, path, domain=mz, type="float32")

# parse it back in
parseAnalyze(path)
```

`parseImzML` *Parse an imzML File*

Description

Parse an imzML file for mass spectrometry (MS) imaging experiment metadata and spectrum-level metadata.

Usage

```
parseImzML(file, ibd = FALSE, extra = NULL,
    extraArrays = NULL, check = ibd, ...)
```

Arguments

`file`	The file path to the ".imzML" file.
`ibd`	Should the binary data file be attached?
`extra`	Additional cvParam or userParam tags to parse from spectrum and/or scan tags by their *accession* or *name* attributes.
`extraArrays`	Additional binary data arrays to parse based on identifying *accession* or *name* cvParam tags.
`check`	Should the UUID, checksum, and size of the binary data file be checked against the corresponding imzML tags and binary data array offsets? This can also be a character vector specifying any combination of "checksum", "uuid", and "filesize" to check.
`...`	Not currently used.

Details

The parse imzML file is returned as a `ImzML` object, which is a list-like structure that can be travered via the standard `$`, `"["`, and `"[["` operators. Child nodes that contain cvParams and userParams will be `imzplist` objects which are also list-like structures that can be traversed the same way.

The spectrum-level metadata is an exception and will be read in selectively and represented as `data.frames` where each row contains the metadata for a specific spectrum. Metadata for `positions`, `mzArrays`, and `intensityArrays` will be parsed. These will be available in `$run$spectrumList`.

If `ibd=TRUE`, the binary data arrays are attached as out-of-memory `matter_list` objects. Uncompressed data arrays are attached as their native binary data types. Compressed data arrays are attached as raw byte arrays.

Value

An object of class `ImzML`.

Author(s)

Kylie A. Bemis

See Also

`ImzMeta`, `writeImzML`

Examples

```
# get the path to an example imzML file
path <- exampleImzMLFile()

# parse the file
p <- parseImzML(path, ibd=TRUE, extra=c(TIC="MS:1000285"))
print(p)

# get the spectra positions
p$run$spectrumList$positions

# get the TIC
p$run$spectrumList$extra

# get the m/z and intensity arrays
p$ibd$mz
p$ibd$intensity
```

`writeAnalyze` *Write an Analyze 7.5 File*

Description

Write an Analyze 7.5 file from a N-dimensional array or a matrix with corresponding pixel/voxel positions.

Usage

```
## S4 method for signature 'array'
writeAnalyze(object, file, positions = NULL, domain = NULL,
    type = "float32", ...)

## S4 method for signature 'matter_arr'
writeAnalyze(object, file, positions = NULL, domain = NULL,
    type = "float32", ...)

## S4 method for signature 'sparse_arr'
writeAnalyze(object, file, positions = NULL, domain = NULL,
    type = "float32", ...)
```

Arguments

`object`	Array-like data of at least 3 dimensions, or matrix-like data with columns corresponding to rows in `positions`.
`file`	The file path to use for writing the ".img" and ".hdr" files.
`positions`	A data frame or matrix of pixel/voxel positions corresponding to the columns of `object`.
`domain`	An optional numeric vector of domain values (e.g., m/z-values).
`type`	The data type using for writing the ".img" file. Allowed values are "int16", "int32", "float32", and "float64".
`...`	Not currently used.

Details

If `domain` is provided (e.g., for m/z-values), then a ".t2m" file will also be written.

Value

`TRUE` if the file was successfully written; `FALSE` otherwise. The output file paths are attached as attributes.

Author(s)

Kylie A. Bemis

Examples

```
# create a toy data cube
set.seed(2023)
nx <- 3
ny <- 3
nmz <- 500
mz <- seq(500, 510, length.out=nmz)
intensity <- replicate(nx * ny, rlnorm(nmz))
dim(intensity) <- c(nmz, nx, ny)
path <- tempfile(fileext=".hdr")

# write it in Analyze 7.5 format
writeAnalyze(intensity, path, domain=mz, type="float32")

# parse it
parseAnalyze(path)
```

`writeImzML` *Write an imzML File*

Description

Write an imzML file with experimental and spectrum-level metadata.

Usage

```
## S4 method for signature 'ImzML'
writeImzML(object, file, positions = NULL, mz = NULL, intensity = NULL,
    mz.type = "float64", intensity.type = "float32", asis = FALSE, ...)

## S4 method for signature 'ImzMeta'
writeImzML(object, file, positions, mz, intensity, ...)
```

Arguments

`object`	An object containing MS imaging metadata.
`file`	The file path to use for writing the ".imzML" file.
`positions`	A data frame or matrix of raster positions where the mass spectra were collected. Replaces any existing positions in `object`.

mz	A numeric vector (for "continuous" format) or list of such vectors (for "processed" format) giving the m/z-values of the mass spectra. Used to write the ".ibd" file if provided.
intensity	A numeric matrix (for "continuous" format) or list of numeric vectors (for "processed" format) giving the intensity values of the mass spectra. Used to write the ".ibd" file if provided.
mz.type, intensity.type	The data types for writing the respective arrays to the ".ibd" file. Allowed types are "int32", "int64", "float32", and "float64".
asis	If TRUE and mz and intensity are both file-backed matter objects, then they are only used to infer the binary metadata for writing the ".imzML" file, and the ".ibd" is *not* written.
...	Arguments passed to the ImzML method.

Details

The ImzML method writes the ".imzML" file based on the provided ImzML object. If mz and intensity are *both* provided, then it *also* writes the associated ".ibd" file. It performs only minimal checking that the required tags exist in the ImzML object. It does *not* validate the XML mapping before writing.

The ImzMeta method requires all of positions, mz, and intensity to write the files.

Value

TRUE if the file was successfully written; FALSE otherwise. This return value should be checked to make sure the operation completed, as most failure cases will yield warnings rather than errors. The output file paths are attached as attributes.

Author(s)

Kylie A. Bemis

See Also

ImzMeta, parseImzML

Examples

```
# get the path to an example imzML file
path <- exampleImzMLFile()

# parse the file
p <- parseImzML(path, ibd=TRUE)
print(p)

# get the spectra and positions
mz <- as.list(p$ibd$mz)
intensity <- as.list(p$ibd$intensity)
positions <- p$run$spectrumList$positions

# write the file back out
path2 <- tempfile(fileext=".imzML")
writeImzML(p, path2, positions=positions,
    mz=mz, intensity=intensity)
```

Index