Package ‘rmzqc’

August 18, 2022

Title Creation, Reading and Validation of 'mzqc' Files

Version 0.1.0

Date 2022-08-15

Description Reads, writes and validates 'mzQC' files. The 'mzQC' format is a standardized file format for the exchange, transmission, and archiving of quality metrics derived from biological mass spectrometry data, as defined by the HUPO-PSI (Human Proteome Organisation - Proteomics Standards Initiative) Quality Control working group. See <https://hupo-psi.github.io/mzQC/> for details.

Imports jsonlite, knitr, methods, ontologyIndex, rmarkdown, R6, R6P, testthat, tools

VignetteBuilder knitr

License MIT + file LICENSE

URL https://github.com/MS-Quality-hub/rmzqc

BugReports https://github.com/MS-Quality-hub/rmzqc/issues

Encoding UTF-8

RoxygenNote 7.2.1

Config/testthat/edition 3

NeedsCompilation no

Author Chris Bielow [aut, cre] (<https://orcid.org/0000-0001-5756-3988>), David Jimenez-Morales [rev] (<https://orcid.org/0000-0003-4356-6461>)

Maintainer Chris Bielow <chris.bielow@bsc.fu-berlin.de>

Repository CRAN

Date/Publication 2022-08-18 12:20:02 UTC

R topics documented:

`check_type` *Checks the value's class type, which should match at least of the types given in any_expected_class_types.*

Description

Checks the value's class type, which should match at least of the types given in any_expected_class_types.

Usage

```
check_type(value, any_expected_class_types, expected_length = 0)
```

Arguments

`value`	A certain value (e.g. a single value, data.frame etc)
`any_expected_class_types`	A vector of valid class types, any of which the @p value should have
`expected_length`	The expected length of value (usually to check if its a single value); 0 (default) indicates that length can be ignored

Examples

```
check_type(1, "numeric", 1)   # TRUE
check_type("1", "numeric", 1) # FALSE
check_type(1, "numeric", 2)   # FALSE
check_type("ABC", "character", 1)             # TRUE
check_type("ABC", "character")                # TRUE
check_type("ABC", "character", 2)             # FALSE
check_type(c("ABC", "DEF"), "character", 2)   # TRUE
check_type(1.1, c("numeric", "double"))    # TRUE
check_type(1.1, c("numeric", "double"), 1) # TRUE
check_type(matrix(1:9, nrow=3), "matrix")   # TRUE
check_type(data.frame(a=1:3, b=4:6), c("something", "data.frame"))   # TRUE
```

`CV_` *Define a Singleton class which can hold a CV dictionary (so we do not have to load the .obo files over and over again)*

Description

Define a Singleton class which can hold a CV dictionary (so we do not have to load the .obo files over and over again)

Define a Singleton class which can hold a CV dictionary (so we do not have to load the .obo files over and over again)

Details

Usage: cv_dict = CV_$new() ## uses 'getCVDictionary()' to populate the singleton cv_2 = CV_$new() ## uses the same data without parsing again

Wherever you need this data, simply re-grab the singleton using 'CV_$new()$data'

Super class

`R6P::Singleton` -> `CV_`

Public fields

`data` Stores the data of the singleton.

Methods

Public methods:

- `CV_$byID()`
- `CV_$clone()`

Method `byID()`**:** A function to retrieve a CV using its ID

Usage:

```
CV_$byID(id)
```

Arguments:

`id` A CV accession, e.g. 'MS:1000560'

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CV_$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`filenameToCV` *For a given filename (e.g. "test.mzML"), check the suffix and translate it to an PSI-MS CV term, e.g. 'MS:1000584'*

Description

The following mapping is currently known: .raw : MS:1000563 ! Thermo RAW format .mzML : MS:1000584 ! mzML format .mzData : MS:1000564 ! PSI mzData format .wiff : MS:1000562 ! ABI WIFF format .pkl : MS:1000565 ! Micromass PKL format .mzXML : MS:1000566 ! ISB mzXML format .yep : MS:1000567 ! Bruker/Agilent YEP format .dta : MS:1000613 ! Sequest DTA format .mzMLb : MS:1002838 ! mzMLb format

Usage

```
filenameToCV(filepath)
```

Arguments

`filepath`	A filename (with optional path)

Details

Falls back to 'MS:1000560 ! mass spectrometer file format' if no match could be found.

Upper/lowercase is ignored, i.e. "mzML == mzml".

Value

A CV term accession as string, e.g. 'MS:1000584'

Examples

```
filenameToCV("test.mZmL")  # MS:1000584
filenameToCV("test.raw")  # MS:1000563
filenameToCV(c("test.raw", "bla.mzML"))
```

---

fromDatatoMzQC — *Allow conversion of plain named lists to mzQC objects*

Description

The plain-R representation of your mzQC objects must be wrapped in an outer list, if your mzQC object representation is already a list because upon detecting lists, this function will call 'class$fromData(element)' for every element.

Usage

```
fromDatatoMzQC(mzqc_class, data)
```

Arguments

`mzqc_class`	Prototype of the class to convert 'data' into
`data`	A datastructure of R lists/arrays as obtained by 'jsonlite::fromJSON()'

Examples

```
data = MzQCcvParameter$new("acc", "myName", "value")
data_recovered = fromDatatoMzQC(MzQCcvParameter, list(jsonlite::fromJSON(jsonlite::toJSON(data))))
data_recovered
```

---

getCVDictionary — *Parse the content of 'psi-ms.obo', 'pato.obo', and 'uo.obo' from the 'rmzqc/cv/' folder as ontology and return their union*

Description

See CV_ class to use this function efficiently.

Usage

```
getCVDictionary()
```

Value

a data.frame with columns 'id', 'name', 'def', 'parents', 'children' (and many more) which contains the CV entries

getCVTemplate — *Fills a MzQCcvParameter object with id(accession) and name. The value (if any) needs to be set afterwards.*

Description

Fills a MzQCcvParameter object with id(accession) and name. The value (if any) needs to be set afterwards.

Usage

```
getCVTemplate(accession, mzcv_dict = CV_$new()$data)
```

Arguments

`accession`	The ID (=accession) of the term in the CV
`mzcv_dict`	A CV dictionary, as obtained by getCVDictionary(); defaults to a singleton, which needs to be filled manually beforehand

Value

An instance of MzQCcvParameter

getDefaultCV — *Returns an MzQCcontrolledVocabulary which points to the PSI-MS CV which is currently shipped with this package*

Description

Returns an MzQCcontrolledVocabulary which points to the PSI-MS CV which is currently shipped with this package

Usage

```
getDefaultCV()
```

`getDefaultCVVersion` *Obtains the current 'data-version' from the MS-CV shipped with this package*

Description

Obtains the current 'data-version' from the MS-CV shipped with this package

Usage

```
getDefaultCVVersion()
```

Examples

```
 getDefaultCVVersion() # "4.1.95"
```

`getQualityMetricTemplate` *Fills a MzQCqualityMetric object with id(accession) and name. The value (if any) and unit (if any) need to be set afterwards.*

Description

Fills a MzQCqualityMetric object with id(accession) and name. The value (if any) and unit (if any) need to be set afterwards.

Usage

```
getQualityMetricTemplate(accession, mzcv_dict = CV_$new()$data)
```

Arguments

`accession`	The ID (=accession) of the term in the CV
`mzcv_dict`	A CV dictionary, as obtained by getCVDictionary(); defaults to a singleton, which needs to be filled manually beforehand

Value

An instance of MzQCqualityMetric

`hasFileSuffix` *Checks if filepath ends in suffix (ignoring lower/upper case differences). If suffix does not start with a '.' it is prepended automatically.*

Description

Checks if filepath ends in suffix (ignoring lower/upper case differences). If suffix does not start with a '.' it is prepended automatically.

Usage

```
hasFileSuffix(filepath, suffix)
```

Arguments

`filepath`	A relative or absolute path to a file, whose suffix is checked
`suffix`	This is the suffix we expect (the '.' is prepended internally if missing)

Value

TRUE if yes, FALSE otherwise

Examples

```
hasFileSuffix("bla.txt", "txt")    # TRUE
hasFileSuffix("bla.txt", ".txt")   # TRUE
hasFileSuffix("bla.txt", ".TXT")   # TRUE
hasFileSuffix("foo", "")           # TRUE
hasFileSuffix("", "")              # TRUE
hasFileSuffix("bla.txt", "doc")    # FALSE
hasFileSuffix("bla.txt", ".doc")   # FALSE
hasFileSuffix("fo", ".doc")        # FALSE
hasFileSuffix("", ".doc")          # FALSE
```

`isUndefined` *Tell if a string is undefined (NA or NULL); If yes, and its required by the mzQC standard, we can raise an error.*

Description

You can pass multiple strings, which are all checked. If **any** of them is undefined, the function returns TRUE

Usage

```
isUndefined(s, ..., verbose = TRUE)
```

Arguments

`s`	A string to be checked for NA/NULL
`...`	More strings to be checked
`verbose`	If TRUE and 's' is NULL/NA, will print the name of the variable which was passed in

Examples

```
isUndefined(NA)       ## TRUE
isUndefined(NULL)     ## TRUE
isUndefined(NA, NULL) ## TRUE
isUndefined("")       ## FALSE
isUndefined("", NA)   ## TRUE
isUndefined(NA, "")   ## TRUE
isUndefined(1)        ## FALSE
myVar = NA
isUndefined(myVar)    ## TRUE, with warning "Variable 'myVar' is NA/NULL!"
```

`isValidMzQC` *Checks validity (= completeness) of mzQC objects - or lists (JSON arrays) thereof*

Description

Note: Returns TRUE for empty lists!

Usage

```
isValidMzQC(x, ...)
```

Arguments

`x`	An mzQC refclass (or list of them), each will be subjected to `isValidMzQC()`
`...`	Ellipsis, for recursive argument splitting

Details

You can pass multiple arguments, which are all checked individually. All of them need to be valid, for TRUE to be returned. The reason for combining both list support for arguments and ellipsis (...) into this function is that JSON arrays are represented as lists and you can simply pass them as a single argument (without the need for do.call()) and get the indices of invalid objects (if any). The ellipsis is useful to avoid clutter, i.e. if (!isValidMzQC(a) || !isValidMzQC(b)) doStuff() is harder to read than if (!isValidMzQC(a,b)) doStuff()

Examples

```
isValidMzQC(MzQCcvParameter$new("MS:4000059"))       # FALSE
isValidMzQC(MzQCcvParameter$new("MS:4000059", "Number of MS1 spectra")) # TRUE
isValidMzQC(list(MzQCcvParameter$new("MS:4000059"))) # FALSE
isValidMzQC(list(MzQCcvParameter$new("MS:4000059", "Number of MS1 spectra"))) # TRUE
isValidMzQC(list(MzQCcvParameter$new("MS:4000059", "Number of MS1 spectra")),
            MzQCcvParameter$new()) # FALSE
```

`MzQCanalysisSoftware-class`
Details of the software used to create the QC metrics

Description

Details of the software used to create the QC metrics

Fields

`accession` Accession number identifying the term within its controlled vocabulary.

`name` Name of the controlled vocabulary term describing the software tool.

`version` Version number of the software tool.

`uri` Publicly accessible URI of the software tool or documentation.

`description` (optional) Definition of the controlled vocabulary term.

`value` (optional) Name of the software tool.

`MzQCbaseQuality-class` *Base class of runQuality/setQuality*

Description

Base class of runQuality/setQuality

Fields

`metadata` The metadata for this run/setQuality

`qualityMetrics` Array of MzQCqualityMetric objects

`MzQCcontrolledVocabulary-class` *A controlled vocabulary document, usually pointing to an .obo file*

Description

A controlled vocabulary document, usually pointing to an .obo file

Fields

`name` Full name of the controlled vocabulary.

`uri` Publicly accessible URI of the controlled vocabulary.

`version` (optional) Version of the controlled vocabulary.

Examples

```
MzQCcontrolledVocabulary$new(
  "Proteomics Standards Initiative Quality Control Ontology",
  "https://github.com/HUPO-PSI/mzQC/blob/master/cv/qc-cv.obo",
  "1.2.0")
```

`MzQCcvParameter-class` *A controlled vocabulary parameter, as detailed in the OBO file*

Description

A controlled vocabulary parameter, as detailed in the OBO file

Fields

`accession` Accession number identifying the term within its controlled vocabulary.

`name` Name of the controlled vocabulary term describing the parameter.

`value` (optional) Value of the parameter.

`description` (optional) Definition of the controlled vocabulary term.

Examples

```
MzQCcvParameter$new("MS:4000070",
                    "retention time acquisition range",
                    c(0.2959, 5969.8172))
isValidMzQC(MzQCcvParameter$new("MS:0000000"))
```

`MzQCDateTime-class` *An mzQC-formatted date+time in ISO8601 format, as required by the mzQC spec doc.*

Description

The format is "%Y-%m-%dT%H:%M:%S".

Fields

`datetime` A correctly formatted date time (use as read-only)

Examples

```
dt1 = MzQCDateTime$new("1900-01-01")  ## yields "1900-01-01T00:00:00"
dt2 = MzQCDateTime$new(Sys.time())
## test faulty input
## errors with 'character string is not in a standard unambiguous format'
try(MzQCDateTime$new('lala'), silent=TRUE)
## test roundtrip conversion from/to JSON
dt2$fromData(jsonlite::fromJSON(jsonlite::toJSON(dt1)))
```

`MzQCinputFile-class` *An inputfile within metadata for a run/setQuality*

Description

An inputfile within metadata for a run/setQuality

Fields

`name` The name MUST uniquely match to a location (specified below) listed in the mzQC file.

`location` Unique file location, REQUIRED to be specified as a URI. The file URI is RECOMMENDED to be publicly accessible.

`fileFormat` An MzQCcvParameter with ’accession’ and ’name’.

`fileProperties` An array of MzQCcvParameter, usually with ’accession’, ’name’ and ’value’. Recommended are at least two entries: a) Completion time of the input file (MS:1000747) and b) Checksum of the input file (any child of: MS:1000561 ! data file checksum type).

`MzQCmetadata-class` *The metadata for a run/setQuality*

Description

The metadata for a run/setQuality

Fields

`label` Unique name for the run (for runQuality) or set (for setQuality).

`inputFiles` Array/list of MzQCinputFile objects

`analysisSoftware` Array/list of MzQCanalysisSoftware objects

`cvParameters` (optional) Array of cvParameters objects

`MzQCmzQC-class` *Root element of an mzQC document*

Description

At least one of runQualities or setQualities MUST be present.

Fields

`version` Version of the mzQC format.

`creationDate` Creation date of the mzQC file.

`contactName` Name of the operator/creator of this mzQC file.

`contactAddress` Contact address (mail/e-mail or phone)

`description` Description and comments about the mzQC file contents.

`runQualities` Array of MzQCrunQuality;

`setQualities` Array of MzQCsetQuality

`controlledVocabularies` Array of CV domains used (obo files)

`MzQCqualityMetric-class` *The central class to store QC information*

Description

The central class to store QC information

Fields

`accession` Accession number identifying the term within its controlled vocabulary.

`name` Name of the controlled vocabulary element describing the metric.

`description` (optional) Definition of the controlled vocabulary term.

`value` (optional) Value of the metric (single value, n-tuple, table, matrix). The structure is not checked by our mzQC implementation and must be handled by the caller

`unit` (optional) Array of unit(s), stored as MzQcvParameter

`MzQCrunQuality-class` *A runQuality object. Use to report metrics for individual runs which are independent of other runs.*

Description

The object is an alias for MzQCbaseQuality.

`MzQCsetQuality-class` *A setQuality object. Use it for metrics which are specific to sets, i.e. only for values which only make sense in the set context and cannot be stored as runQuality (see mzQC spec doc).*

Description

The object is an alias for MzQCbaseQuality.

NULL_to_charNA — *Converts a NULL to NA_character_; or returns the argument unchanged otherwise*

Description

This is useful for missing list elements (which returns NULL), but when the missing element in refClass should be NA_character_ (and NULL would return an error)

Usage

```
NULL_to_charNA(char_or_NULL)
```

Arguments

char_or_NULL	A string or NULL

Examples

```
NULL_to_charNA(NA)   ## NA
NULL_to_charNA(NULL) ## NA_character_
NULL_to_charNA("hi") ## "hi"
```

NULL_to_NA — *Converts a NULL to NA; or returns the argument unchanged otherwise*

Description

This is useful for missing list elements (which returns NULL), but when the missing element in refClass should be NA (and NULL would return an error)

Usage

```
NULL_to_NA(var_or_NULL)
```

Arguments

var_or_NULL	A variable of any kind or NULL

Examples

```
NULL_to_NA(NA)   ## NA
NULL_to_NA(NULL) ## NA
NULL_to_NA("hi") ## "hi"
```

parseOBO	*Get the information of each CV term from an obo file.*

Description

Get the information of each CV term from an obo file.

Usage

```
parseOBO(cv_obo_file)
```

Arguments

`cv_obo_file`	A path to an .obo file

Value

A data.frame containing CV term information

removeFileSuffix	*Removes the last suffix (including the last dot) from a filename. If no dot exists, the full string is returned.*

Description

Removes the last suffix (including the last dot) from a filename. If no dot exists, the full string is returned.

Usage

```
removeFileSuffix(filepath)
```

Arguments

`filepath`	A filename (with optional path – which is retained)

Value

The input with removed suffix

Examples

```
removeFileSuffix("test.tar.gz")  # --> 'test.tar'
removeFileSuffix("test.mzML")  # --> 'test'
removeFileSuffix("/path/to/test.mzML")  # --> '/path/to/test'
removeFileSuffix("test_no_dot")  # --> 'test_no_dot'
```

rmzqc — *rmzqc: A package for reading, validating, and writing mzQC files.*

Description

The core function of the package is reading mzQC files into an RefClasses wrapped data structure and writing such data to file again.

toAnalysisSoftware — *From an ID, e.g. "MS:1003162" (for PTX-QC), and some additional information, create an 'analysisSoftware' node for mzQC*

Description

From an ID, e.g. "MS:1003162" (for PTX-QC), and some additional information, create an 'analysisSoftware' node for mzQC

Usage

```
toAnalysisSoftware(id, version = "unknown", uri = NULL, value = NA_character_)
```

Arguments

`id`	The CV accession
`version`	The version of the tool which created the metric/mzQC
`uri`	URI to the homepage, or if NULL (default), will be extracted from the definition in the PSI MS-CV (if possible)
`value`	An optional name for the software (if different from the CV's name)

Value

An MzQCanalysisSoftware object

Examples

```
toAnalysisSoftware(id = "MS:1003162", version = "1.0.13")
```

`toQCMetric` *Create an 'MzQCqualityMetric' object from two inputs*

Description

Create an 'MzQCqualityMetric' object from two inputs

Usage

```
toQCMetric(id, value, on_violation = c("error", "warn"))
```

Arguments

`id`	The CV accession
`value`	The data, as computed by some QC software in the required format.
`on_violation`	What to do when 'value' is not of the correct type (according to the given 'id')? Default: "error"; or "warn"

Details

The inputs are:

- an ID of a QC metric, e.g. "MS:4000059" (number of MS1 spectra)
- a value

The value must be in the correct format depending on the metric. The value type (see below) is checked (a warning/error is given if mismatching): The following requirements for values apply:

- single value: R single value; the unit is obtained from the CVs 'has_units'
- n-tuple: an R vector, e.g. using c(1,2,3), i.e. all values have the same type; the unit is obtained from the CVs 'has_units'
- table: an R data.frame(); all columns defined using CVs 'has_column' must be present (a warning/error is given otherwise)
- matrix: an R matrix, i.e. all values have the same type; the unit is obtained from the CVs 'has_units'

Upon violation, an error (default) or a warning is emitted:

```
toQCMetric(id = "MS:4000059", value = data.frame(n = 1)) # errors: wrong value format
```

Value

An MzQCanalysisSoftware object

Examples

```
toQCMetric(id = "MS:4000059", value = 13405) # number of MS1 spectra
```

`writeMZQC` *Writes a full mzQC object to disk.*

Description

The filename should have an '.mzQC' as suffix (warning otherwise).

Usage

```
writeMZQC(filepath, mzqc_obj)
```

Arguments

`filepath`	A filename (with path) to write to.
`mzqc_obj`	An mzQC object, which is serialized to JSON and then written to disk

Index