Package ‘rmzqc’

April 16, 2024

Title Creation, Reading and Validation of 'mzqc' Files

Version 0.5.4

Date 2024-04-15

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, jsonvalidate, 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

Config/testthat/edition 3

RoxygenNote 7.3.1

NeedsCompilation no

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

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

Repository CRAN

Date/Publication 2024-04-16 07:50:03 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_` *CV_*

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)

Details

Get the full data by calling the 'getData()' function (which returns a list containing a 'CV', 'URI' and 'version'), or 'getCV()' which is a shorthand for 'getData()$CV'. You can set your own custom CV by calling 'setData()'. By default, the latest release of the PSI-MS-CV (see `getCVDictionary`). Wherever you need this data, simply re-grab the singleton using 'CV_$new()' (or use the convenience function getCVSingleton() from outside the package)

Super class

`R6P::Singleton` -> `CV_`

Methods

Public methods:

- `CV_$ensureHasData()`
- `CV_$byID()`
- `CV_$setData()`
- `CV_$getData()`
- `CV_$getCV()`
- `CV_$clone()`

Method **`ensureHasData()`:** Make sure that the CV data is loaded

Usage:

`CV_$ensureHasData()`

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

Usage:

`CV_$byID(id)`

Arguments:

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

Method **`setData()`:** Set a user-defined object (= a list of 'CV', 'URI' and 'version'), as obtained from `getCVDictionary`

Usage:

`CV_$setData(cv_data)`

Arguments:

`cv_data` The result of a call to `getCVDictionary`

Method **`getData()`:** Gets the underlying data (CV, URI and version)

Usage:

`CV_$getData()`

Method **`getCV()`:** A shorthand for 'getData()$CV', i.e. the CV data.frame.

Usage:

`CV_$getCV()`

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.

Examples

```
## Not run:
  cv_dict = CV_$new() ## uses 'getCVDictionary()' to populate the singleton
  cv_2 = CV_$new() ## uses the same data without parsing again
  cv_2$setData(getCVDictionary("custom", "https://my.com/custom.obo"))

## End(Not run)
```

`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 of R objects (from jSON) to mzQC objects*

Description

Allow conversion of plain named lists of R objects (from jSON) to mzQC objects

Usage

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

Arguments

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

Examples

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

`fromDatatoMzQCobj` *Allow conversion of a plain R object (obtained from jSON) to an mzQC object*

Description

If you have a list of elements, call fromDatatoMzQC.

Usage

```
fromDatatoMzQCobj(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 = fromDatatoMzQCobj(MzQCcvParameter, jsonlite::fromJSON(jsonlite::toJSON(data)))
data_recovered
```

`getCVDictionary` *Fetch and parse the 'psi-ms.obo' and some metadata from the usual sources to use as ontology.*

Description

If `use_local_fallback` is TRUE, this function will never fail. Otherwise, it may fail if the internet connection is flawed or internal URLs related to GitHubs API become stale.

Usage

```
getCVDictionary(
  source = c("latest", "local", "custom"),
  custom_uri = NULL,
  use_local_fallback = TRUE
)
```

Arguments

`source`	Where to get the PSI-MS CV from: - 'latest' will download 'psi-ms.obo' from https://api.github.com/repos/HUPO-PSI/psi-ms-CV/releases/latest - 'local' will use rmzqc/cv/psi-ms.obo' (which might be outdated, if you need the latest terms) - 'custom' uses a user-defined URI in 'custom_uri'
`custom_uri`	Used when 'source' is set to 'custom'. The URI can be local or remote, e.g. 'c:/obo/my.obo' or 'https://www.abc.com/my.obo'
`use_local_fallback`	When downloading a file from a URI fails, should we fall back to the local psi-ms.obo shipped with rmzqc?

Details

A 'pato.obo', and 'uo.obo' from the 'rmzqc/cv/' folder are automatically merged into the result.

See CV_ class to use this function efficiently.

Value

A list with 'CV', 'URI' and 'version', where 'CV' is a data.frame with columns 'id', 'name', 'def', 'parents', 'children' (and many more) which contains the CV entries

---

`getCVInfo`	*Returns an* `MzQCcontrolledVocabulary` *for the currently used CV (see* `getCVSingleton`*) using* `getCVSingleton()$getData()$URI` *and* `$version`*.*

Description

Returns an `MzQCcontrolledVocabulary` for the currently used CV (see `getCVSingleton`) using `getCVSingleton()$getData()$URI` and `$version`.

Usage

```
getCVInfo()
```

`getCVSingleton`	*Returns the CV singleton. See* `CV_`*.*

Description

Returns the CV singleton. See `CV_`.

Usage

```
getCVSingleton()
```

`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, CV = getCVSingleton())
```

Arguments

`accession`	The ID (=accession) of the term in the CV
`CV`	A CV dictionary, as obtained by getCVDictionary(); defaults to the global singleton, which is populated automatically

Value

An instance of MzQCcvParameter

`getDefaultCV` *Returns an MzQCcontrolledVocabulary for the currently used CV (see `getCVSingleton`)*

Description

Returns an MzQCcontrolledVocabulary for the currently used CV (see `getCVSingleton`)

Usage

```
getDefaultCV()
```

Note

This function will be deprecated soon. Use `getCVInfo` instead.

`getLatest_PSICV_URL` *Get the latest PSI-MS CV release URL*

Description

This may fail (e.g. if no internet connection is available, or URLs became invalid) then 'NULL' will be returned instead of an URL. A warning may be emitted, if the URL is out of date (i.e. the GitHub API changed).

Usage

```
getLatest_PSICV_URL()
```

`getLocal_CV_Version` *Obtains the 'data-version' from a local (i.e. non-url) PSI-MS-CV*

Description

Obtains the 'data-version' from a local (i.e. non-url) PSI-MS-CV

Usage

```
getLocal_CV_Version(local_PSIMS_obo_file)
```

Arguments

`local_PSIMS_obo_file`	A path to a local file, e.g. 'c:/temp/my.obo'

Examples

```
getLocal_CV_Version(system.file("./cv/psi-ms.obo", package="rmzqc")) # "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, CV = getCVSingleton())
```

Arguments

`accession`	The ID (=accession) of the term in the CV
`CV`	A CV dictionary, as obtained by getCVDictionary(); defaults to the global singleton, which is populated automatically

Value

An instance of MzQCqualityMetric

`getSyntaxValidator` *Get a syntax validator for mzQC*

Description

Get a syntax validator for mzQC

Usage

```
getSyntaxValidator()
```

`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
```

`localFileToURI` *Convert a local filename, e.g. "./myData/test.mzML" to a proper URI (e.g. "file:///user/bielow/myData/test.mzML")*

Description

Relative filenames are made absolute. Backslashes as path separators are replaced by forward slashes (as commonly seen on Windows).

Usage

```
localFileToURI(local_filename, must_exist = TRUE)
```

Arguments

`local_filename`	Path to a file (can be relative to current getwd(); or absolute)
`must_exist`	Require the file to exist

Value

A URI starting with "file:///" followed by an absolute path

`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/psi-ms-CV/releases/download/v4.1.129/psi-ms.obo",
  "4.1.129")
```

`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, see `toQCMetric`.

`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 local path to an .obo file

Value

A data.frame containing CV term information

`readMZQC` *Read a JSON file in mzQC format into an MzQCmzQC root object*

Description

Read a JSON file in mzQC format into an MzQCmzQC root object

Usage

```
readMZQC(filepath)
```

Arguments

`filepath`	A filename (with path) to read from.

Value

An MzQCmzQC root object from which all the data can be extracted/manipulated

---

`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'
```

`removeIfExists` *Remove a file, if it exists (useful for temporary files which may or may not have been created)*

Description

Remove a file, if it exists (useful for temporary files which may or may not have been created)

Usage

```
removeIfExists(tmp_filename)
```

Arguments

`tmp_filename`	A path to a local file

Value

NULL if file is missing, otherwise TRUE/FALSE depening on successful removal

---

`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

```
# use 'version = packageVersion("PTXQC")' if the package is installed
toAnalysisSoftware(id = "MS:1003162", version = "1.0.12")
```

`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
```

`validateFromFile` *Syntactically validates an mzQC document which is present as a file.*

Description

The returned TRUE/FALSE has additional attributes in case of errors. Use attributes(result) to access them.

Usage

```
validateFromFile(filepath, verbose = TRUE)
```

Arguments

`filepath`	A path to a file (e.g. "c:/my.mzQC", or "test.mzQC")
`verbose`	Show extra information if validation fails

Value

TRUE/FALSE if validation was successful/failed

`validateFromObj` *Syntactically validates an mzQC document which is already in memory as mzQC root object, as obtained by, e.g. readMZQC().*

Description

This method is less performant than validateFromString, because it needs to convert the R object to a JSON string first.

Usage

```
validateFromObj(mzqc_root, verbose = TRUE)
```

Arguments

`mzqc_root`	An mzQC root object
`verbose`	Show extra information if validation fails

Details

The returned TRUE/FALSE has additional attributes in case of errors. Use attributes(result) to access them.

Value

TRUE/FALSE if validation was successful/failed

`validateFromString` *Syntactically validates an mzQC document which is already in memory as JSON string. e.g. the string "{ mzQC : {}}"*

Description

If the string object passed into this function contains multiple elements (length > 1). then they will be concatenated using '\n' before validation.

Usage

```
validateFromString(JSON_string, verbose = TRUE)
```

Arguments

`JSON_string`	A string which contains JSON (multiple lines allowed)
`verbose`	Show extra information if validation fails

Details

The returned TRUE/FALSE has additional attributes in case of errors. Use attributes(result) to access them.

Value

TRUE/FALSE if validation was successful/failed

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

Description

You can in theory also provide any mzQC subelement, but the resulting mzQC file will not validate since its incomplete.

Usage

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

Arguments

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

Details

The filename should have '.mzQC' (case sensitive) as suffix. There will be a warning otherwise.

Index