Package ‘EML’

February 27, 2021

Type Package
Title Read and Write Ecological Metadata Language Files
Description Work with Ecological Metadata Language (‘EML’) files. ‘EML’ is a widely used metadata standard in the ecological and environmental sciences, described in Jones et al. (2006), <doi:10.1146/annurev.ecolsys.37.091305.110031>.
Version 2.0.5
License MIT + file LICENSE
BugReports https://github.com/ropensci/EML/issues
Encoding UTF-8
LazyData true
Depends R (>= 2.10)
Suggests knitr, taxadb, tibble, testthat, covr, EML, units, htmlwidgets, shiny, shinyjs, spelling
Imports xml2, methods, digest, emld (>= 0.5.0), jqr, jsonlite, uuid, rmarkdown, utils, dplyr
VignetteBuilder knitr
RoxygenNote 7.1.1
Language en-US
NeedsCompilation no
Author Carl Boettiger [aut, cre, cph]
   (https://orcid.org/0000-0002-1642-628X), Matthew B. Jones [aut] (https://orcid.org/0000-0003-0077-4738), Mitchell Maier [ctb] (https://orcid.org/0000-0001-6955-0535), Bryce Mecum [ctb] (https://orcid.org/0000-0002-0381-3766), Maëlle Salmon [ctb] (https://orcid.org/0000-0002-2815-0399), Jeanette Clark [ctb] (https://orcid.org/0000-0003-4703-1974)
Maintainer Carl Boettiger <cboettig@gmail.com>
Repository CRAN
Date/Publication 2021-02-27 06:00:07 UTC
### Description

builds factor table for shiny app

### Usage

```r
build_factors(att_table, data)
```

### Arguments

- **att_table**  
  (data.frame) input attributes table
- **data**  
  (data.frame) input data
### build_units_table

**Description**
builds unit table for shiny app

**Usage**
```
build_units_table(in_units, eml_units)
```

**Arguments**
- `in_units`: input units
- `eml_units`: eml units

### detect_delim

**Description**
Automatically detect line delimiters in a text file

This helper function was written expressly for `set_physical` to be able to automate its `recordDelimiter` argument.

**Usage**
```
detect_delim(path, nchar = 1000)
```

**Arguments**
- `path`: (character) File to search for a delimiter
- `nchar`: (numeric) Maximum number of characters to read from disk when searching

**Value**
(character) If found, the delimiter, it not, \n
### eml

**Description**
eml

**Format**
A list with constructor functions
eml_get

**Description**

eml_get

**Usage**

eml_get(x, element, from = "list", ...)

**Arguments**

- **x**: an EML object or child/descendant object
- **element**: name of the element to be extracted. If multiple occurrences are found, will extract all
- **from**: explicit type for the input format. Possible values: "xml", "json", "list", or "guess" with "list" as the default.
- **...**: additional arguments

**Examples**

```r
f <- system.file("tests", emld::eml_version(), "eml-datasetWithUnits.xml", package = "emld")
eml <- read_eml(f)
eml_get(eml, "physical")
eml_get(eml, "attributeList")

## The first argument need not be an "eml" class, it could be a child element; e.g.
eml_get(eml$dataset$dataTable, "physical")
```

eml_validate

**Description**

eml_validate processes an EML document using the XSD schema for the appropriate version of EML and determines if the document is schema-valid as defined by the XSD specification

**Usage**

eml_validate(eml, encoding = "UTF-8", schema = NULL)
Arguments

- `eml`: file path, xml_document,
- `encoding`: optional encoding for files, default UTF-8.
- `schema`: path to schema

Value

Whether the document is valid (logical)

Note

this function is simply an alias to `eml_validate` in `emld` package

Examples

```r
f <- system.file("extdata", "example.xml", package = "emld")

## validate file directly from disk:
eml_validate(f)

## validate an eml object:
eml <- read_eml(f)
eml_validate(eml)
```

Description

get_attributes

Usage

```r
get_attributes(x, eml = NULL)
```

Arguments

- `x`: an "attributeList" element from an emld object
- `eml`: The full eml document, needed only if <references> outside of attributes must be resolved.
get_numberType

Details

EML metadata can use "references" elements which allow one attribute to use metadata declared elsewhere in the document. This function will automatically resolve these references and thus infer the correct metadata.

Value

a data frame whose rows are the attributes (names of each column in the data file) and whose columns describe metadata about those attributes. By default separate tables are given for each type

Examples

```r
f <- system.file("tests", emld::eml_version(),
"eml-datasetWithAttributeValueMethods.xml", package = "emld")
eml <- read_eml(f)
get_attributes(eml$dataset$ dataTable$attributeList)
```

get_numberType 
Get EML numberType

Description

returns the EML numberType (either 'real', 'integer', 'whole', or 'natural') of input values

Usage

generateNumberType(values)

Arguments

values (numeric/character) a vector of values, if vector is non-numeric will return NA

Value

the numberType of values (either 'real', 'integer', 'whole', or 'natural').

Examples

```r
## Not run:
# To get numberType for each column in a data.frame:
unlist(lapply(df, function(x) get_numberType(x)))
## End(Not run)
```
get_unitList

Description

get_unitList

Usage

get_unitList(x = NULL)

Arguments

x

an emld object

Details

If no unitList is provided, the function reads in the eml-unitDictionary defining all standard units and unitTypes. This provides a convenient way to look up standard units and their EML-recognized names when defining metadata, e.g. in the table passed to `set_attributes()`.

Value

a list with two data.frames: "units", a table defining unit names, types, and conversions to SI, and "unitTypes", defining the type of unit. For instance, the unit table could define "Hertz" as a unit of unitType frequency, and the unitType define frequency as a type whose dimension is 1/time.

Examples

# Read in additional units defined in a EML file

f <- system.file("tests", emld::eml_version(),
    "eml-datasetWithUnits.xml",
    package = "emld"
)
eml <- read_eml(f)
unitList <- get_unitList(eml)

## Read in the definitions of standard units:
get_unitList()
get_unit_id

Description

A function to assist with getting valid EML unit ids (see examples). Warning: ensure returned unit is equivalent to input unit (for example "pH" will return "picohenry" which may or may not be equivalent to the input unit "pH").

Usage

get_unit_id(input_units, eml_version = emld::eml_version())

Arguments

input_units (character|vector) input units that needs valid EML unit ids
eml_version (character) the eml schema version desired (there is a change in the way eml units are named from eml-2.1.1 to eml-2.2.0)

Value

(character) A valid EML unit id. If no valid EML unit id can be found, the function will output a warning, along with a preformatted custom unit id.

Examples

## Not run:
# The following all return the same id
get_unit_id("kilometersPerSquareSecond")
get_unit_id("kilometerPerSecondSquared")
get_unit_id("Kilometers per seconds squared")
get_unit_id("km/s^2")
get_unit_id("km s-2")
get_unit_id("s-2 / kilometers-1") # this works but is not advised

## End(Not run)

htmlwidgets_attributes

Launch attributes htmlwidget

Description

Used to call handsontable html widget to build attributes
is_standardUnit

Usage

htmlwidgets_attributes(df, type = NULL)

Arguments

df (data.frame) the data.frame of data that needs an attribute table
type (character) either "attributes", "units", or "factors"

is_standardUnit

Usage

is_standardUnit(x)

Arguments

x name of unit to check

Value

TRUE if unit is exact match to the id of a unit in the Standard Units Table, FALSE otherwise.

Examples

is_standardUnit("amperePerMeter") # TRUE
is_standardUnit("speciesPerSquareMeter") # FALSE

read_eml

Description

Read an EML file into R as an emld object.

Usage

read_eml(x, from = "xml")
Arguments

- `x`: path to an EML file
- `from`: explicit type for the input format. Possible values: "xml", "json", "list", or "guess" with "xml" as the default.

Value

an emld object (list / S3 object)

Examples

```r
f <- system.file("extdata", "example.xml", package = "emld")
eml <- read_eml(f)
```

Description

`set_attributes`

Usage

```r
set_attributes(
  attributes,
  factors = NULL,
  col_classes = NULL,
  missingValues = NULL
)
```

Arguments

- `attributes`: a joined table of all attribute metadata
- `factors`: a table with factor code-definition pairs; see details
- `col_classes`: optional, list of R column classes ("ordered", "numeric", "factor", "Date", or "character", case sensitive) will let the function infer missing "domain" and "measurementScale" values for attributes column. Should be in same order as attributeNames in the attributes table, or be a named list with names corresponding to attributeNames in the attributes table.
- `missingValues`: optional, a table with missing value code-definition pairs; see details
Details

The attributes data frame must use only the recognized column headers shown here. The attributes data frame must contain columns for required metadata. These are:

For all data:

• attributeName (required, free text field)
• attributeDefinition (required, free text field)
• measurementScale (required, "nominal", "ordinal", "ratio", "interval", or "dateTime", case sensitive) but it can be inferred from col_classes.
• domain (required, "numericDomain", "textDomain", "enumeratedDomain", or "dateTimeDomain", case sensitive) but it can be inferred from col_classes.

For numeric (ratio or interval) data:

• unit (required). Unitless values should use "dimensionless" as the unit.

For character (textDomain) data:

• definition (required)

For dateTime data:

• formatString (required)

Other optional allowed columns in the attributes table are: source, pattern, precision, numberType, missingValueCode, missingValueCodeExplanation, attributeLabel, storageType, minimum, maximum

The factors data frame, required for attributes in an enumerated domain, must use only the following recognized column headers:

• attributeName (required)
• code (required)
• definition (required)

The missingValues data frame, optional, can be used in the case that multiple missing value codes need to be set for the same attribute. This table must contain the following recognized column headers.

• attributeName (required)
• code (required)
• definition (required)

Value

an eml "attributeList" object
set_coverage

Description

set_coverage

Usage

set_coverage(
    beginDate = character(),
    endDate = character(),
    date = character(),
    sci_names = character(),
    geographicDescription = character(),
    westBoundingCoordinate = numeric(),
    eastBoundingCoordinate = numeric(),
    northBoundingCoordinate = numeric(),
    southBoundingCoordinate = numeric(),
    altitudeMinimum = numeric(),
    altitudeMaximum = numeric(),
    altitudeUnits = character()
)

Arguments

beginDate Starting date for temporal coverage range.
endDate   End date for temporal coverage range
date      give a single date, or vector of single dates covered (instead of beginDate and
          endDate)
sci_names string (space separated) or list or data frame of scientific names for species cov-
          ered. See details
geographicDescription text string describing the geographic location
westBoundingCoordinate Decimal longitude for west edge bounding box
eastBoundingCoordinate Decimal longitude for east edge bounding box
northBoundingCoordinate Decimal latitude value for north of bounding box
southBoundingCoordinate Decimal latitude value for south edge of bounding box
altitudeMinimum    minimum altitude covered by the data (optional)
altitudeMaximum    maximum altitude covered by the data (optional)
altitudeUnits      name of the units used to measure altitude, if given
set_coverage provides a simple and concise way to specify most common temporal, taxonomic, and geographic coverage metadata. For certain studies this will not be well suited, and users will need the more flexible but more verbose construction using "new()" methods; for instance, to specify temporal coverage in geological epoch instead of calendar dates, or to specify taxonomic coverage in terms of other ranks or identifiers.

Value

a coverage object for EML

Note

If "sci_names" is a data frame, column names of the data frame are rank names. For user-defined "sci_names", users must make sure that the order of rank names they specify is from high to low. Ex. "Kingdom","Phylum","Class","Order","Family","Genus","Species","Common"

Examples

coverage <-
  set_coverage(
    begin = "2012-06-01", end = "2013-12-31",
    sci_names = "Sarracenia purpurea",
    geographicDescription = "California coast, down through Baja, Mexico",
    west = -122.44, east = -117.15,
    north = 37.38, south = 30.00
  )

set_methods

Description

set_methods

Usage

set_methods(
  methods_file,
  instrumentation = character(),
  software = NULL,
  sampling_file = NULL,
  sampling_coverage = NULL,
  sampling_citation = NULL,
  qualityControl_file = NULL
)
**Arguments**

- `methods_file` Path to a file (markdown or .docx) containing a description of the methods used
- `instrumentation` optional, text describing instrumentation used in methods
- `software` optional, an EML software node describing software used in methods
- `sampling_file` optional, Path to a file (.md or .docx) describing sampling method
- `sampling_coverage` optional, coverage node for methods, e.g. set_coverage()
- `sampling_citation` optional, a citation element describing the sampling protocol
- `qualityControl_file` optional, path to a file (.md or .docx) describing quality control methods

**Value**

A methods object

**Examples**

```r
f <- system.file("examples/hf205-methods.md", package = "EML")
set_methods(methods_file = f)
```

```r
## Can also import from methods written in a .docx MS Word file.
f <- system.file("examples/hf205-methods.docx", package = "EML")
set_methods(methods_file = f)
```

**Description**

Will calculate the file size, checksum, and checksum authentication method algorithm automatically if the argument `objectName` is a file that exists.

**Usage**

```r
set_physical(
  objectName,
  id = character(),
  numHeaderLines = character(),
  numFooterLines = character(),
  recordDelimiter = detect_delim(objectName),
  fieldDelimiter = ",",
  collapseDelimiters = logical(),
)```
Arguments

**objectName**  
name for the object, usually a filename like "hf205-1.csv"

**id**  
optional, an id value for the `<physical>` element in EML, for use in referencing

**numHeaderLines**  
Number of header lines preceding data. Lines are determined by the physicalLineDelimiter, or if it is absent, by the recordDelimiter. This value indicated the number of header lines that should be skipped before starting to parse the data.

**numFooterLines**  
Number of footer lines following data. Lines are determined by the physicalLineDelimiter, or if it is absent, by the recordDelimiter. This value indicated the number of footer lines that should be skipped after parsing the data. If this value is omitted, parsers should assume the data continues to the end of the data stream.

**recordDelimiter**  
This element specifies the record delimiter character when the format is text. The record delimiter is usually a linefeed (`\n`) on UNIX, a carriage return (`\r`) on MacOS, or both (`\r\n`) on Windows/DOS. Multiline records are usually delimited with two line ending characters, for example on UNIX it would be two linefeed characters (`\n\n`). As record delimiters are often non-printing characters, one can use either the special value "\n" to represent a linefeed (ASCII 0x0a) and "\r" to represent a carriage return (ASCII 0x0d). Alternatively, one can use the hex value to represent character values (e.g., 0x0a).

**fieldDelimiter**  
"." character by default (for csv files). This element specifies a character to be used in the object for indicating the ending column for an attribute. The delimiter character itself is not part of the attribute value, but rather is present in the column following the last character of the value. Typical delimiter characters include commas, tabs, spaces, and semicolons. The only time the fieldDelimiter character is not interpreted as a delimiter is if it is contained in a quoted string (see quoteCharacter) or is immediately preceded by a literalCharacter. Non-printable quote characters can be provided as their hex values, and for tab characters by its ASCII string "\t". Processors should assume that the field starts in the column following the previous field if the previous field was fixed, or in the column following the delimiter from the previous field if the previous field was delimited.
collapseDelimiters
The collapseDelimiters element specifies whether sequential delimiters should
be treated as a single delimiter or multiple delimiters. An example is when a
space delimiter is used; often there may be several repeated spaces that should
be treated as a single delimiter, but not always. The valid values are yes or no.
If it is set to yes, then consecutive delimiters will be collapsed to one. If set to
no or absent, then consecutive delimiters will be treated as separate delimiters.
Default behavior is no; hence, consecutive delimiters will be treated as separate
delimiters, by default.

literalCharacter
This element specifies a character to be used for escaping special character val-
ues so that they are treated as literal values. This allows "escaping" for special
characters like quotes, commas, and spaces when they are intended to be used in
an attribute value rather than being intended as a delimiter. The literalCharacter is typically a \\.

quoteCharacter
This element specifies a character to be used in the object for quoting values so
that field delimiters can be used within the value. This basically allows delimiter
"escaping". The quoteCharacter is typically a '" or '. When a processor encounters
a quote character, it should not interpret any following characters as a delimiter
until a matching quote character has been encountered (i.e., quotes come in
pairs). It is an error to not provide a closing quote before the record ends. Non-
printable quote characters can be provided as their hex values.

attributeOrientation
Specifies whether the attributes described in the physical stream are found in
columns or rows. The valid values are column or row. If set to 'column', then
the attributes are in columns. If set to 'row', then the attributes are in rows. Row
orientation is rare.

size
This element contains information of the physical size of the entity, by default represented in bytes unless the sizeUnit attribute is provided to change the units.

sizeUnit
the unit in which size is measured; default is 'bytes'

authentication
This element describes authentication procedures or techniques, typically by
giving a checksum value for the object. The method used to compute the au-
thentication value (e.g., MD5) is listed in the method attribute.

authMethod
the method for authentication checksum, e.g. MD5

characterEncoding
This element contains the name of the character encoding. This is typically
ASCII or UTF-8, or one of the other common encodings.

encodingMethod
This element lists a encoding method used to encode the object, such as base64,
BinHex.

compressionMethod
This element lists a compression method used to compress the object, such as
zip, compress, etc. Compression and encoding methods must be listed in the
order in which they were applied, so that decompression and decoding should
occur in the reverse order of the listing. For example, if a file is compressed
using zip and then encoded using MIME base64, the compression method would
be listed first and the encoding method second.
set_responsibleParty

url optional. The complete url from which the data file can be downloaded, if possible.

Value

an EML physical object, such as used in a dataTable element to define the format of the data file.

Examples

set_physical("hf205-01-TPexp1.csv")
# FIXME set recordDelimiter based on user's system?
# FIXME richer distribution options? use set_distribution at top level?

Description

set_responsibleParty

Usage

set_responsibleParty(
  givenName = NULL,
  surName = NULL,
  organizationName = NULL,
  positionName = NULL,
  address = NULL,
  phone = NULL,
  electronicMailAddress = NULL,
  onlineUrl = NULL,
  userId = NULL,
  id = NULL,
  email = NULL
)

Arguments

givenName individual’s given names (list or vector for multiple names). OR a person object.
surName individual name
organizationName if party is an organization instead of an individual, name for the org
positionName individual’s position, i.e. "Researcher", "Graduate Student", "Professor"
address address object, see 'eml$address' to build an address object
phone individual or organization phone number
electronicMailAddress email address (alternatively, can use 'email' argument)
onlineUrl  a URL to the homepage of the individual or organization
userId   the user’s ID, usually within a particular system (KNB, DataONE)
id     Identifier for this block, ideally an ORCID id (optional)
email     alias for electronicMailAddress

Value

A emld object for any responsibleParty (e.g. creator, contact, etc)

Examples

carl <- set_responsibleParty(as.person("Carl Boettiger <cboettig@ropensci.org>"))
matt <- set_responsibleParty("Matthew", "Jones", email = "mbjones@nceas.ucsb.edu")

---

Description

set_software

Usage

set_software(codemeta)

Arguments

codemeta  codemeta object, see examples

Value

an eml software element

Examples

cm <- jsonlite::read_json(system.file("extdata/codemeta.json", package = "EML"))
software <- set_software(cm)
my_eml <- eml$eml(packageId = "eml-1.2", system = "knb", software = software)

# write_eml(my_eml, "test.xml")
set_taxonomicCoverage

Description

set_taxonomicCoverage

Usage

set_taxonomicCoverage(sci_names, expand = FALSE, db = "itis")

Arguments

sci_names  string (space separated) or list or data frame of scientific names for species covered.
expand     Set to TRUE to use `[taxadb]` to expand sci_names into full taxonomic classifications
db         The taxonomic database to query (when expand is set to TRUE). See `[taxadb::filter_name]` for valid options. Defaults to `itis`.

Details

Turn a data.frame or a list of scientific names into a taxonomicCoverage block sci_names can be a space-separated character string or a data frame with column names as rank name or a list of user-defined userDefinedTaxonomicClassification

Value

a taxonomicCoverage object for EML

Note

If "sci_names" is a data frame, column names of the data frame are rank names. For user-defined "sci_names", users must make sure that the order of rank names they specify is from high to low. Ex. "Kingdom","Phylum","Class","Order","Family","Genus","Species","Common" EML permits any rank names provided they go in descending order.

Examples

sci_names <- data.frame(
  Kingdom = "Plantae",
  Phylum = "Phaeophyta",
  Class = "Phaeophyceae",
  Order = "Laminariales",
  Family = "Lessoniaceae",
  Genus = "Macrocystis",
  Species = "pyrifer"
taxon_coverage <- set_taxonomicCoverage(sci_names)

# Examples that may take > 5s

## use a list of lists for multiple species
sci_names <- list(list(
    Kindom = "Plantae",
    Phylum = "Phaeophyta",
    Class = "Phaeophyceae",
    Order = "Laminariales",
    Family = "Lessoniaceae",
    Genus = "Macrocystis",
    Species = "pyrifera"
))
set_taxonomicCoverage(sci_names)

---

**set_TextType**

**set_TextType**

---

**Description**

For any EML element of class TextType, this function can be used to generate the appropriate EML from a markdown-formatted file.

**Usage**

`set_TextType(file = NULL, text = NULL)`

**Arguments**

- `file` path to a file providing formatted input text, see details.
- `text` a plain text character string which will be used directly as the content of the node if no file is given

**Details**

If the ‘rmarkdown’ package is installed, then the input file can be a Microsoft Word (.docx) file, a markdown file, or other file recognized by Pandoc (see https://pandoc.org), which will automate the conversion to a docbook. Otherwise, the input file should already be in docbook format (with .xml or .dbk extension). Note that pandoc comes pre-installed in RStudio and is required for the rmarkdown package.

**Value**

a TextType object that can be coerced into any element inheriting from TextType, see examples
Examples

```r
## using a simple character string
a <- set_TextType(text = "This is the abstract")

## Using an external markdown file
f <- system.file("examples/hf205-abstract.md", package = "EML")
a <- set_TextType(f)

## Can also import from methods written in a .docx MS Word file.
f <- system.file("examples/hf205-abstract.docx", package = "EML")
a <- set_TextType(f)

## Documents with title headings use `section` instead of `para` notation
f <- system.file("examples/hf205-methods.docx", package = "EML")
d <- set_TextType(f)
```

---

**Description**

Define custom units, including new unitTypes. Note that it is not necessary to define most common units.

**Usage**

```r
set_unitList(units, unitTypes = NULL, as_metadata = FALSE)
```

**Arguments**

- `units`: a data.frame describing the custom units, see details.
- `unitTypes`: optional, a data.frame defining any additional unitTypes not already defined
- `as_metadata`: logical, default FALSE. If true, returns an `additionalMetadata` element, see below.

**Details**

The units data.frame must have the following columns: - id: the referenced name of unit (singular), e.g. 'meter', 'second' - unitType: the base type of unit, e.g. 'length'. If not from a standard type, a new unitType must be provided - multiplierToSI: the multiplicative constant to convert to the SI unit. - parentSI: the name of the parent SI unit, e.g. second. - description: a text string describing the unit of measure. The following columns are optional: - name: usually the same as the id of the unit, e.g. `name = "second"` - abbreviation: common abbreviation, e.g. `s` - constantToSI: an additive constant to convert to the equivalent SI unit. If not given, default is "0"
In practice, researchers may save these tables of custom units they frequently use in an external .csv or other format and read them in to R for ready re-use.

The `unitType` table must have the following columns: - id: the name by which the `unitType` is referred to. - name: optional, default is same as the id - dimension: name of a base dimension of the unit - power: the power to which the dimension is raised (NA implies power of 1)

**Value**

unitList list object

**Examples**

```r
## create the "unitType" table for custom unit
id <- c("speed", "speed", "acceleration", "acceleration", "frequency")
dimension <- c("length", "time", "length", "time", "time")
power <- c(NA, "-1", NA, "-2", "-1")
unitTypes <- data.frame(id = id, dimension = dimension, power = power, stringsAsFactors = FALSE)

## Create the units table
id <- c("minute", "centimeter")
unitType <- c("time", "length")
parentSI <- c("second", "meter")
multiplierToSI <- c("0.0166", "1")
description <- c("one minute is 60 seconds", "centimeter is 1/100 of a meter")
units <- data.frame(id = id, unitType = unitType, parentSI = parentSI, multiplierToSI = multiplierToSI, description = description, stringsAsFactors = FALSE)

unitList <- set_unitList(units, unitTypes)
```

---

**shiny_attributes**

Create/Edit EML attributes

**Description**

Create/edit EML attributes, custom units, and factors in a shiny environment.

**Usage**

```r
shiny_attributes(data = NULL, attributes = NULL)
```

**Arguments**

- **data** (data.frame) the data.frame of data that needs an attribute table
- **attributes** (data.frame) an existing attributes table
Details

Attributes can be created from scratch using `shiny_attributes()`. Or an existing attribute table can be edited using `shiny_attributes(NULL, attributes)`. Or new attributes can be created from a data table using `shiny_attributes(data, NULL)`. If attributes are created from a data table, fields such as 'attributeName' and 'numberType' will be automatically completed based on the attributes within the data table. If both existing attributes and data table are entered (i.e. `shiny_attributes(data, attributes)`), any automatically generated fields based attributes within the data table **will not** override any non-empty fields in the entered attributes.

Examples

```r
## Not run:
# from scratch
out <- shiny_attributes(NULL, NULL)

# from data
data <- iris
out <- shiny_attributes(data, NULL)

# from existing attributes
file <- system.file("tests", emld::eml_version(),
  "eml-datasetWithAttributelevelMethods.xml",
  package = "emld"
)
exml <- read_eml(file)
x <- exml$dataset$dataTable$attributeList
df <- get_attributes(x, exml)
out <- shiny_attributes(NULL, df$attributes)

# from attributes and data
out <- shiny_attributes(data, df$attributes)
```

## End(Not run)

---

**table_to_r**

**handsontable to r**

Description

Takes a handsontable and converts to r data.frame for shiny app

Usage

`table_to_r(table)`

Arguments

- `table` input table
write_eml

Description

write_eml

Usage

write_eml(eml, file, namespaces = NULL, ns = "eml", ...)

Arguments

eml an emld class object
file file name to write XML.
namespaces named character vector of additional XML namespaces to use.
ns root namespace abbreviation
... additional arguments to write_xml

Value

If file is not specified, the result is a character string containing the resulting XML content. Otherwise return silently.

Examples

f <- system.file("extdata", "example.xml", package = "emld")
eml <- read_eml(f)
write_eml(eml, "test.xml")
eml_validate("test.xml")
unlink("test.xml") # clean up
Index

* datasets
  eml, 3

build_factors, 2
build_units_table, 3
detect_delim, 3
eml, 3
eml_get, 4
eml_validate, 4
get_attributes, 5
get_numberType, 6
get_unit_id, 8
get_unitList, 7

htmlwidgets_attributes, 8

is_standardUnit, 9

read_eml, 9

set_attributes, 10
set_coverage, 12
set_methods, 13
set_physical, 3, 14
set_responsibleParty, 17
set_software, 18
set_taxonomicCoverage, 19
set_TextType, 20
set_unitList, 21
shiny_attributes, 22


table_to_r, 23

write_eml, 24
write_xml, 24