Package 'dmdScheme'

August 22, 2022

Title Domain Specific MetaData Scheme

Description Forms the core for developing own domain specific metadata schemes.
It contains the basic functionality needed for all metadata schemes based on the
'dmdScheme'. See R.M. Krug and O.L. Petchey (2019) <DOI:10.5281/zenodo.3581970>.

Version 1.3.5

Date 2022-08-22

URL https://UZH-PEG.github.io/dmdScheme/,
https://github.com/UZH-PEG/dmdScheme/

BugReports https://github.com/UZH-PEG/dmdScheme/issues/

Depends R (>= 3.5.0)

Imports tools, methods, utils, magrittr (>= 1.5), readxl (>= 1.2.0),
xml2, rlang (>= 0.3.1), rmarkdown, knitr, digest (>= 0.6),
writexl, rappdirs, yaml, stringr

License MIT + file LICENSE

Encoding UTF-8

StagedInstall true

RoxygenNote 7.2.1

Suggests covr (>= 3.2.1), testthat (>= 2.0.1), here (>= 0.1), shiny,
openxlsx, xfun

VignetteBuilder knitr

NeedsCompilation no

Author Rainer M. Krug [aut, cre] (<https://orcid.org/0000-0002-7490-0066>),
Owen L. Petchey [ctb] (<https://orcid.org/0000-0002-7724-1633>)

Maintainer Rainer M. Krug <Rainer.Krug@uzh.ch>

Repository CRAN

Date/Publication 2022-08-22 07:30:02 UTC

R topics documented:

as_dmdScheme . . . 3
as_dmdScheme_raw . . . 4
as_eml . . . 5
as_xml . . . 6
as_xml_list . . . 7
cache . . . 8
cat_ln . . . 8
dmdScheme . . . 9
dmdScheme_example . . . 10
dmdScheme_raw . . . 10
format_dmdScheme_xlsx . . . 11
lookup_tokens . . . 11
make_example . . . 12
make_index . . . 13
make_new_package . . . 14
new_dmdScheme_validation . . . 15
open_new_spreadsheet . . . 16
pkg.dmdScheme . . . 17
print.dmdSchemeData . . . 18
print.dmdSchemeSet . . . 18
print.dmdScheme_validation . . . 19
print_dmdScheme_validation_details . . . 21
print_dmdScheme_validation_summary . . . 21
read_excel . . . 22
read_excel_raw . . . 23
read_xml . . . 24
report . . . 24
run_app . . . 26
scheme_active . . . 27
scheme_make . . . 30
scheme_path_index_template . . . 31
scheme_path_xlsx . . . 31
scheme_path_xml . . . 32
toTranspose . . . 32
upgrade_old_files . . . 33
valErr_extract . . . 34
valErr_info . . . 34
valErr_isOK . . . 35
valErr_TextErrCol . . . 35
validate . . . 36
validateAllowedValues . . . 37
validateDataFileMetaDataDataFileExists . . . 37
validateIDField . . . 38
validateStructure . . . 38
validateSuggestedValues . . . 39
validateTypes . . . 39

write_excel 40
write_xml 40

Index **42**

`as_dmdScheme` *Generic function to convert the data stored in the object* `x` *into a new object of class* `dmdScheme...`

Description

Generic function to convert the data stored in the object `x` into a new object of class `dmdScheme...`

Usage

```
as_dmdScheme(x, keepData = FALSE, ..., verbose = FALSE)

## S3 method for class 'dmdSchemeData_raw'
as_dmdScheme(
  x,
  keepData = TRUE,
  convertTypes = TRUE,
  warnToError = TRUE,
  checkVersion = TRUE,
  ...,
  verbose = FALSE
)

## S3 method for class 'dmdSchemeSet_raw'
as_dmdScheme(
  x,
  keepData = FALSE,
  warnToError = TRUE,
  convertTypes = TRUE,
  checkVersion = TRUE,
  ...,
  verbose = FALSE
)

## S3 method for class 'xml_document'
as_dmdScheme(x, keepData = TRUE, useSchemeInXml = NULL, ..., verbose = FALSE)
```

Arguments

`x`	object to be converted
`keepData`	if the data should be kept or replaced with one row with NAs
`...`	additional arguments for methods

verbose	give verbose progress info. Useful for debugging.
convertTypes	if TRUE, the types specified in the types column are used for the data type. Otherwise, they are left at type character
warnToError	if TRUE, warnings generated during the conversion will raise an error
checkVersion	if TRUE, a version mismatch between the package and the data x will result in an error. If FALSE, the check will be skipped.
useSchemeInXml	if TRUE, use scheme definition in xml and raise an error if the xml does not contain a scheme definition. If False, use the scheme definition from the corresponding installed package, even if the xml contains a scheme definition. if NULL (the default), use the definition in the xml if it contains a definition, if not use the corresponding definition from the installed package.

Value

dmdScheme as object of class dmdScheme_set

Examples

```
as_dmdScheme(dmdScheme_raw(), keepData = TRUE)
as_dmdScheme(dmdScheme_raw()$Experiment)

xml <- as_xml(dmdScheme_example())
x <- as_dmdScheme(xml)
all.equal(dmdScheme_example(), x)
```

---

as_dmdScheme_raw *Generic function to convert the data stored in the object* x *into a new object of class* dmdScheme_raw...

Description

Generic function to convert the data stored in the object x into a new object of class dmdScheme_raw...

Usage

```
as_dmdScheme_raw(x, ...)

## S3 method for class 'dmdSchemeData'
as_dmdScheme_raw(x, ...)

## S3 method for class 'dmdSchemeSet'
as_dmdScheme_raw(x, ...)

## S3 method for class 'xml_document'
as_dmdScheme_raw(x, useSchemeInXml = NULL, ...)
```

Arguments

x	object to be converted
...	additional arguments for methods
useSchemeInXml	if TRUE, use scheme definition in xml and raise an error if the xml does not contain a scheme definition. If False, use the scheme definition from the corresponding installed package, even if the xml contains a scheme definition. if NULL (the default), use the definition in the xml if it contains a definition, if not use the corresponding definition from the installed package.

Value

dmdScheme as object of class dmdScheme_raw

Examples

```
as_dmdScheme_raw(dmdScheme(), keepData = TRUE)
```

---

as_eml *Generic function to convert an object to an object which can be saved as EML*

Description

Generic function to convert an object to an object which can be saved as EML

Usage

```
as_eml(x, ...)

## Default S3 method:
as_eml(x, ...)
```

Arguments

x	object to be converted.
...	additional arguments for methods

Value

A list object which can be converted to EML. It can be written to a file using write_eml(). The resulting eml file can be validated using eml_validate().

NB: This does only validate the EML format, and NOT the validation as defined in the scheme definition package!

as_xml *Generic function to convert an object to xml*

Description

Generic function to convert an object to xml

Usage

```
as_xml(x, output = "metadata", ...)

## S3 method for class 'dmdSchemeData'
as_xml(x, output = "metadata", ...)

## S3 method for class 'dmdSchemeSet'
as_xml(x, output = "metadata", ...)
```

Arguments

x	object to be converted.
output	specifies the content and format of the exported xml.
	"metadata" : export of the metadata only with no format attributes
	"complete" : export tof the complete sheme, i.e. "metadata" plus the scheme definition. This is a self contained format which contains all attributes.
...	additional arguments for methods

Value

an xml_document object

Examples

```
x <- as_xml( dmdScheme_example() )
x

## returns \code{xml_document} object
```

`as_xml_list` *Generic function to convert an object to a list containing xml(s)*

Description

Generic function to convert an object to a list containing xml(s)

Usage

```
as_xml_list(x, output = "metadata", ...)

## S3 method for class 'dmdSchemeSet'
as_xml_list(x, output = "metadata", ...)
```

Arguments

`x`	object to be converted.
`output`	specifies the content and format of the exported xml.
	"metadata" : export of the metadata only with no format attributes
	"complete" : export tof the complete sheme, i.e. "metadata" plus the scheme definition. This is a self contained format which contains all attributes.
`...`	additional arguments for methods

Value

a `list()` where each element is `xml_document` object

Examples

```
x <- as_xml_list( dmdScheme_example() )
x

## returns a \code{list()} with one \code{xml_document} object
```

cache *Return cache directory*

Description

If the cache folder does not exist, and `createPermanent = FALSE`, a temporary location is used. To make the cache permanent, call

Usage

```
cache(..., delete = FALSE, createPermanent = FALSE)
```

Arguments

`...`	sub caches
`delete`	if `TRUE`, the cache directory will be deleted.
`createPermanent`	if `TRUE`, the folder will be created. This is done, when `delete = TRUE` after deleting the directory.

Details

```
cache(createPermanent = TRUE)
```

and restart your R session.

Value

fully qualified path to the cache folder

cat_ln *cat with linefeed at the end Title*

Description

Copied from tibble:::cat_line()

Usage

```
cat_ln(...)
```

Arguments

`...`	will be handed over to `cat(..., "\n")`

dmdScheme	*Object of class* dmdSchemeSet *containing the authorative definition of the dmdScheme.*

Description

The dataset contains the authorative definition of the dmdScheme. It contains no data, except one row NA. There are two S3 classes defined and used:

dmdSchemeData**:** a data.frame with the class dmdSchemeData. Each column is one property of the metadata. It has the following attributes:

propertyName: name of the data. In the spreadsheet, it is in the cell below (in the Experiment tab) or to the right (other tabs).

unit: the unit of the data in each column.

type: the type of the data in the column. These will be validated in the validate function. See there for details.

suggestedValues: suggested values for the data of each column. These will be validated in the validate function. See there for details.

allowedValues: allowed values for the data of each column. These will be validated in the validate function. See there for details.

Description: general description of the columns.

names: the names of the columns.

dmdSchemeSet: a list with where each element is a dmdSchemeData object with additional attributes:

propertyName: name of the dmdScheme used. In the spreadsheet, it is in the cell H1 in ther in the Experiment tab (DATA dmeScheme v0.9.9.)

dmdSchemeVersion: version of the dmdScheme used. In the spreadsheet, it is in the cell H1 in ther in the Experiment tab (DATA dmeScheme v0.9.9.)

names: the names of the dmdSchemeData sets. In the spreadsheet, the names of the tabs.

Usage

```
dmdScheme()
```

Examples

```
dmdScheme()
```

dmdScheme_example *Object of class* dmdSchemeSet *containing example data.*

Description

The dataset contains example data. It was created by using the code below.

Usage

```
dmdScheme_example()
```

Examples

```
dmdScheme_example()
```

dmdScheme_raw *Object of class* dmdScheme_raw *containing the raw data as read in.*

Description

The dataset contains raw data. An object of class dmdScheme_raw is returned by the function read_excel_raw with the argument raw = TRUE and read_excel_raw. It is usually an intermediate object, as in the normal workflow, this object is automatically converted to an object of class dmdSchemeSet.

Usage

```
dmdScheme_raw()
```

Details

dmdSchemeData_raw: a data.frame as returned by the function read_excel with the class dmdSchemeData_raw

dmdSchemeSet_raw: a list with where each element is a dmdSchemeData_raw object with additional attributes:

- **propertyName:** name of the dmdScheme for which this object contains the raw data. In the spreadsheet, it is in the cell H1 in ther in the Experiment tab (DATA dmeScheme v0.9.9.)
- **dmdSchemeVersion:** version of the dmdScheme used. In the spreadsheet, it is in the cell H1 in ther in the Experiment tab (DATA dmeScheme v0.9.9.)
- **names:** the names of the dmdSchemeData_raw sets. In the spreadsheet, the names of the tabs.

Examples

```
dmdScheme_raw()
```

format_dmdScheme_xlsx *Format the metadata scheme file*

Description

Loads fn_org), formats it and saves it as fn_new.

Usage

```
format_dmdScheme_xlsx(fn_org, fn_new, keepData = TRUE)
```

Arguments

fn_org	file name of the original excel file to be formated
fn_new	file name where the final xlsx should be saved to. If missing, it will not be saved.
keepData	if TRUE, data from data cells will be empty

Value

invisibly the workbook as a workbook object as created by xlsx.crerateWorkbook()

lookup_tokens *Replace tokens with vlues from a dmdScheme*

Description

For a detailed explanation of these tokens see the vignette Create and Customize the index Template.

Usage

```
lookup_tokens(tokens, scheme, author = "")
```

Arguments

tokens	a character vector containing tokens. These can be enclosed in %%TOKEN%% or not.
scheme	a dmdSchemeSet object
author	the author of the index document

Value

a list of the length of the input vector tokens containing the objects returned by the tokens. Null if the tokens contains invalid values.

Examples

```
lookup_tokens(
  tokens = c(
      "%%Treatments.*.2%%",
     "%%Experiment.*.*%%",
     "Measurement.method.3"
   ),
  scheme = dmdScheme_example(),
)
```

`make_example` *Create examples in working directory*

Description

Each package based on a `dmdScheme` can contain examples. This function is the interface to these examples. In the package `dmdScheme`, no examples are included. The function has two basic usages:

1. by using `make_example(schemeName = "NameOfTheScheme")` all included examples are listed
2. by using `make_example(name = "basic", schemeName = "NameOfTheScheme")` it will create the example named `basic` in a subdirectory in the current working directory. An existing directory with the same name, will nod be overwritten!

Usage

```
make_example(name)
```

Arguments

`name`	name of the example

Details

The examples have to be located in a directory called `example_data`. The function is doing two things:

1. Copying the **complete** directory from the `example_data` directory to the current working directory
2. running `knitr::purl` on **all** `./code/*.Rmd` to extract the code into `.R` script files. If you want to include an RMarkdown files in the `./code` directory from thisa, use the `.rmd` extension (small letters).

Value

invisibly `NULL`

Examples

```
make_example()
## Not run:
make_example("basic")

## End(Not run)
```

`make_index` *Generic function to create the* `index.md` *file to accompany the data deposit package*

Description

Generic function to create the `index.md` file to accompany the data deposit package

Title

Usage

```
make_index(
  scheme,
  path = ".",
  overwrite = FALSE,
  template = scheme_path_index_template(),
  author = NULL,
  make = c("html", "pdf"),
  pandoc_bin = "pandoc",
  pandoc_args = "-s",
  ...
)

## S3 method for class 'dmdSchemeSet'
make_index(
  scheme,
  path = ".",
  overwrite = FALSE,
  template = scheme_path_index_template(),
  author = NULL,
  make = c("pdf", "html"),
  pandoc_bin = "pandoc",
  pandoc_args = "-s",
  ...
)
```

Arguments

`scheme`	a dmdScheme from which the values for the tokens in the `template` should be taken

path	path to where the ‘index‘ should be created. The file name of the created index is identical to the file name of the template.
overwrite	if TRUE, the target index file will be overwritten automatically, unless the target index is equal to the template, in which case, an error would be raised in all cases.
template	template to be used for the index file. For details see the vignette **Create and Customize the index template**. The default template is at `system.file("index.md", package = "dmdScheme")`
author	of the index file
make	character vector containing types into which the generated index file should be converted to. default is html and pdf. **This function uses pandoc for the conversion!.**
pandoc_bin	pandoc executable. Needs fully qualified path when not in `$PATH`.
pandoc_args	arguments for calling pandoc
...	additional arguments for methods

Value

returns path to the created `index.md` file

Examples

```
## Not run:
  # takes to long for CRAN
  make_index( dmdScheme_example(), path = tempdir() )

## End(Not run)
```

`make_new_package` *Create anew package skelleton to add functiuonality to the currently active scheme.*

Description

This function is not for the user of a scheme, but for the development process of a new scheme.

Usage

```
make_new_package(path = ".")
```

Arguments

path	path where the package should be created. Default is the current working directory.

Details

A new metadata scheme can be created by using the function `scheme_make`. This function will create a package to add functionality to the currently used scheme as a package which will depend on `scheme_active()`. This function uses the function `package.skeleton()` from the `utils` package to create a new directory for the new metadata scheme, and adds a function `aaa.R` which loads the current package whenever the new package is loaded as well as some fields to the `DESCRIPTION` file.

For a documentation of the workflow to create a new scheme, see the vignette **Howto Create a new scheme**.

Value

invisibly `NULL`

Examples

```
make_new_package(
  path = tempdir()
)
```

---

new_dmdScheme_validation

Create new `dmdScheme_validation` *object*

Description

Create new `dmdScheme_validation` object

Usage

```
new_dmdScheme_validation()
```

Value

new `dmdScheme_validation` object

`open_new_spreadsheet` *Open the metadata scheme as a spreadsheet in a spreadsheet editor*

Description

Open `system.file(paste0(schemeName, ".xlsx"), package = schemeName)` in excel. New data can be entered and the file has to be saved at a different location as it is a read-only file.

Usage

```
open_new_spreadsheet(
  file = NULL,
  open = TRUE,
  keepData = FALSE,
  format = TRUE,
  overwrite = FALSE,
  verbose = FALSE
)
```

Arguments

`file`	if not `NULL`, the template will be saved to this file.
`open`	if `TRUE`, the file will be opened. This can produce different results depending on the OS, browsr and browser settings.
`keepData`	if `TRUE` the data entry areas will be emptied. If `FALSE`. the example data will be included.
`format`	if `FALSE` the sheet will be opened as the sheet is. if `TRUE`, it will be formated nicely.
`overwrite`	if `TRUE`, the file specified in `file` will be overwritten. if `FALSE`, an error will be raised ehen the file exists.
`verbose`	give verbose progress info. Useful for debugging.

Value

invisibly the fully qualified path to the file which **would** have been opened, if `open == TRUE`.

Examples

```
## Not run:
  open_new_spreadsheet(schemeName = "dmdScheme", format = FALSE, verbose = TRUE)

## End(Not run)
```

`pkg.dmdScheme` *dmdScheme: A package containing the framework for Domain specific MetaData Schemes*

Description

Metadata is essential for the managing and archiving of data. Consequentially, metadata schemes, which standardise the property names used in specifying the metadata, play an essential role in this. Nevertheless (or because of this), metadata schemes are usually big, complex, difficult to read and understand, and, in consequence, are not used as often as they should be.

Details

This package provides a framework called `dmdScheme`, which

- **makes it easier to develop a domain specific metadata scheme**: by using a spreadsheet as the base for defining the new scheme
- **provides basic functionality for the new metadata scheme**: including entering, validating, exporting and saving of the new metadata
- **makies it easier to enter new metadata**: by uising a spreadsheet to enter the new metadata which can than be imported and exported as xml

This package provides for the **creator** of a new Domain Specific MetaData Scheme:

1. a simple way of discussing and developing a new scheme as it is represented in a spreadsheet
2. a function to create a new R package for a new domain specific metadata scheme which is based on a spreadsheet containing the definition and inherits all the functionality of this package (validation, printing, export, import, ...)
3. easy update of the new scheme based by re-importing the new version of the scheme from the spreadsheet by simply calling one function
4. easy extension of the functionality as the whole architecture is based on S3 methods and the new scheme inherits from the `dmdScheme` objects.

This package provides for the **user** of a Domain Specific MetaData Scheme:

1. the authorative definition of an example `dmdScheme`
2. object definitions for this scheme for R
3. Excel spreadsheet for entering the metadata for an experiment
4. functions to validate this metadata
5. functions to export the metadata to xml files, one per data file

print.dmdSchemeData *Print method for* dmdSchemeData *object*

Description

Print method for dmdSchemeData object

Usage

```
## S3 method for class 'dmdSchemeData'
print(
  x,
  ...,
  printAttr = TRUE,
  printExtAttr = FALSE,
  printData = TRUE,
  .prefix = ""
)
```

Arguments

x	object of type dmdSchemeSet
...	additional arguments - not used here
printAttr	default TRUE - attributes are printed
printExtAttr	default FALSE - additional attributes are not printed (e.g. class)
printData	default TRUE - data is printed
.prefix	mainly for internal use - prefix for all printed lines

Value

invisibly x

print.dmdSchemeSet *Print method for* dmdSchemeSet *object*

Description

Print method for dmdSchemeSet object

Usage

```
## S3 method for class 'dmdSchemeSet'
print(
  x,
  ...,
  printAttr = TRUE,
  printExtAttr = FALSE,
  printData = TRUE,
  .prefix = ""
)
```

Arguments

x	object of type dmdSchemeSet
...	additional arguments - not used here
printAttr	default TRUE - attributes are printed
printExtAttr	default FALSE - additional attributes are not printed (e.g. class)
printData	default TRUE - data is printed
.prefix	mainly for internal use - prefix for all printed lines

Value

invisibly x

---

print.dmdScheme_validation

Print method for dmdScheme_validation *object*

Description

When using different values for format, different outputs are generated:

- "default"print x as list
- "summary"print the description and errors of x as structured output, using the format as specified in the argument format
- "details"print the details of x as structured output, using the format as specified in the argument format

Usage

```
## S3 method for class 'dmdScheme_validation'
print(
  x,
  level = 1,
  listLevel = 3,
```

```
  type = "default",
  format = "markdown",
  error = c(0, 1, 2, 3, NA),
  ...
)
```

Arguments

x	object of class dmdScheme_validation
level	level at which the header structure should start
listLevel	level at which the elements should be represented as lists and not headers anymore
type	type of output, can be either "default", "summary" or "details". Default is "default"
format	format in which the details tables should be printed. All values as used in knitr::kable() are allowed.
error	numeric v ector, containing error levels to print. Default is all error levels.
...	additional arguments for the function knitr::kable() function to format the table.

Value

invisibly returns x

Examples

```
x <- validate(dmdScheme_raw())

## default printout as list
x

## the same as
print(x, type = "default")

## the summary
print(x, type = "summary")

## and the details
print(x, type = "details")

## can be used in a Rmd file like:
# ```{r, results = "asis"}
#    print(result, level = 2, listLevel = 20, type = "summary")
# ```
```

print_dmdScheme_validation_details

Internal function to ptint dmdScheme_validation of format `summary`

Description

Internal function to ptint dmdScheme_validation of format `summary`

Usage

```
print_dmdScheme_validation_details(
  x,
  level,
  listLevel,
  format,
  error = c(0, 1, 2, 3, NA),
  ...
)
```

Arguments

`x`	as in print.dmdScheme_validation
`level`	as in print.dmdScheme_validation
`listLevel`	as in print.dmdScheme_validation
`format`	as in print.dmdScheme_validation
`error`	numeric v ector, containing error levels to print. Default is all error levels.
`...`	as in print.dmdScheme_validation

Value

as in print.dmdScheme_validation

print_dmdScheme_validation_summary

Internal function to ptint dmdScheme_validation of format `summary`

Description

Internal function to ptint dmdScheme_validation of format `summary`

Usage

```
print_dmdScheme_validation_summary(
  x,
  level,
  listLevel,
  error = c(0, 1, 2, 3, NA)
)
```

Arguments

x	as in print.dmdScheme_validation
level	as in print.dmdScheme_validation
listLevel	as in print.dmdScheme_validation
error	numeric v ector, containing error levels to print. Default is all error levels.

Value

as in print.dmdScheme_validation

---

read_excel *Read scheme data from Excel file into* dmdSchemeSet *object*

Description

Reads the data from an Excel file. Validation of the scheme version and scheme name is always done. Additional validations are done depending on the arguments `validate`. See detals below.

Usage

```
read_excel(
  file,
  keepData = TRUE,
  verbose = FALSE,
  raw = FALSE,
  validate = TRUE
)
```

Arguments

file	the name of the Excel file (.xls or .xlsx) containing the data to be read.
keepData	if TRUE, the data in the spreadsheet `file` will be kept (as in dmdScheme_example. If FALSE, it will be replaced with one row with NAs as in dmdScheme. Only used when raw == FALSE.
verbose	give verbose progress info. Useful for debugging.

raw	if TRUE the imported spreadsheet file will be returned as an object of class dmdScheme_raw. If FALSE, it will be converted to an dmdScheme object.
validate	if TRUE results are validated using validate(validateData = FALSE, errorIfStructFalse = TRUE).Consequently, an error is raised if the resulting scheme can not be successfully validated against the one in the package. There are not many cases where you want to change this value to FALSE. But if you do, the result will not be validated. **This can lead to invalid schemes!**.

Value

either if raw = TRUE a list of data.frames from the worksheets of Class dmdScheme_raw, otherwise an object of class dmdSchemeSet

Examples

```
fn <- scheme_path_xlsx()
read_excel(
  file = fn
)

read_excel(
  file = fn,
  raw = TRUE
)
```

---

read_excel_raw *Read scheme data from Excel file into dmdScheme_raw object*

Description

Reads the data from an Excel file as is and no validation. Only validation of the scheme version and scheme name is done (when checkVersion = TRUE).

Usage

```
read_excel_raw(file, verbose = FALSE, checkVersion = TRUE)
```

Arguments

file	the name of the Excel file (.xls or .xlsx) containing the data to be read.
verbose	give verbose progress info. Useful for debugging.
checkVersion	if TRUE, check for version or scheme conflicts between the package scheme version and the scheme version of the file. Aborts with an error if there is a conflict.

Value

object of class dmdScheme_raw

Examples

```
read_excel_raw( scheme_path_xlsx() )
```

`read_xml` *Function to read x from an XML file*

Description

Read the XML file `file` and convert it to a `dmdScheme` object using the function `as_dmdScheme()`.

Usage

```
read_xml(file, keepData = TRUE, useSchemeInXml = NULL, verbose = FALSE)
```

Arguments

`file`	Path to file or connection to write to.
`keepData`	if the data should be kept or replaced with one row with NAs
`useSchemeInXml`	if `TRUE`, use scheme definition in xml and raise an error if the xml does not contain a scheme definition. If False, use the scheme definition from the corresponding installed package, even if the xml contains a scheme definition. if `NULL` (the default), use the definition in the xml if it contains a definition, if not use the corresponding definition from the installed package.
`verbose`	give verbose progress info. Useful for debugging.

Examples

```
# write_xml(dmdScheme_raw(), file = tempfile())
```

`report` *Generic function for creating a report from an object* `x`

Description

This generic function creates a report based on the object.

Usage

```
report(
  x,
  file = tempfile(),
  open = TRUE,
  report = "html",
  report_author = "Myself",
  report_title = "Report of something",
  ...
)

## S3 method for class 'character'
report(
  x,
  file = tempfile(),
  open = TRUE,
  report = "html",
  report_author = "Tester",
  report_title = "Validation of data against dmdScheme",
  ...
)

## S3 method for class 'dmdScheme_validation'
report(
  x,
  file = tempfile(),
  open = TRUE,
  report = "html",
  report_author = "Tester",
  report_title = "Validation of data against dmdScheme",
  ...
)
```

Arguments

x	object of which the report should be creted used to select a method
file	name of the file containing the generated report, including extension. If missing, ot will be saved as a temporary file in the temporary folder.
open	if TRUE, open the report. Default: TRUE
report	determines if and in which format a report of the validation should be generated. Allowed values are: • none: no report is generated • html: a html (.html) report is generated and opened • pdf : a pdf (.pdf) report is generated and opened • word: a word (.docx)report is generated and opened Additional values can be implemented by the different methods and will be documented in the Details section.

report_author	name of the author to be included in the report
report_title	title of the report to be included in the report
...	further arguments passed to or from other methods

Details

report.character creates a report of the object returnes from a validate().

report.dmdScheme_validation creates a report of the object returnes from a validate().

Value

return the path and filename of the report

Methods (by class)

- report(character): report of a dmdScheme_validation object.
- report(dmdScheme_validation): report of a dmdScheme_validation object.

Examples

```
## Not run:
## This examples requires pandoc
## Report of `dmdScheme_validation`
report( scheme_path_xlsx() )

## End(Not run)


## Report of `dmdScheme_validation`
## Not run:
# This needs pandoc to run successfully
report( validate(dmdScheme_raw()) )
report(
   x = dmdScheme_raw(),
   report = "html",
   report_author = "The Author I am",
   report_title = "A Nice Report"
)

## End(Not run)
```

---

run_app *Run shiny app.*

Description

The shiny app allows the entering, validating, and exporting of the metadata without using R. See https://deanattali.com/2015/04/21/r-package-shiny-app/

Usage

```
run_app()
```

Value

return value from `runApp()`

Examples

```
## Not run:
run_app()

## End(Not run)
```

`scheme_active` *Functions to manage schemes*

Description

`scheme_active()`**:** Shows the name and version of the active scheme.

`scheme_default()`**:** Shows the name of the default scheme which comes with the package and can not be deleted. If `name`and `version` is specified, the default scheme to be used will be set. **There is no need to do this only internally!**. Otherwise, the scheme repository used is only returned.

`scheme_download()`**:** Scheme definitions can be stored in an online repo. The default is a github repo at `https://github.com/Exp-Micro-Ecol-Hub/dmdSchemeRepository`. This function dowloads a scheme definition, specified by `name` and `version`, and saves it locally under the name `destfile`

`scheme_install()`**:** Installed schemes are copied to `cache("installedSchemes")` and , if necessary, an `.xlsx` definition is saved in addition. These can be listed by using scheme_list.

`scheme_install_r_package()`**:** Install R package for scheme `name` `version` definition using the script `install_R_package.R` in the scheme package.

`scheme_installed()`**:** Checks if a scheme is installed

`scheme_list()`**:** Lists all definitions for schemes which are installed. Each follows the pattern `SCHEMENAME_SCHEMEVERSION.EXT`. All files with the same basename but different extensions represent different representations of the same scheme definition and are effectively equivalent, only that the tab Documentation can only be found in the `.xls` files.

`scheme_list_in_repo()`**:** Scheme definitions can be stored in an online repo.

`scheme_repo()`**:** Get or set scheme repository. If `repo` is specified, the scheme repository to be used is set. Otherwise, the scheme repository used is only returned.

`scheme_uninstall()`**:** Installed schemes are deleted from `cache("installedSchemes")` and moved to a temporary folder which is rteturned invisibly.

`scheme_use()`**:** Switch from the current scheme to a new scheme as defined in the scheme `schemeDefinition`. Installed schemes can be listed by using `scheme_list()`. New schemes can be added to the library via a call to `scheme_add()`. The name of the active scheme is saved in `dmdScheme_active`

Usage

```
scheme_active()

scheme_default(name = NULL, version = NULL)

scheme_download(
  name,
  version,
  destfile = NULL,
  overwrite = FALSE,
  baseurl = scheme_repo(),
  ...
)

scheme_install(
  name,
  version,
  repo = scheme_repo(),
  file = NULL,
  overwrite = FALSE,
  install_package = FALSE
)

scheme_install_r_package(name, version, reinstall = FALSE)

scheme_installed(name, version)

scheme_list()

scheme_list_in_repo(baseurl = scheme_repo(), ...)

scheme_repo(repo = NULL)

scheme_uninstall(name = NULL, version = NULL)

scheme_use(name = NULL, version = NULL)
```

Arguments

name	a character string containing the name of the scheme definition
version	a character string containing the version of the scheme definition
destfile	a character string containing the name of the downloaded scheme definition. If NULL, a temporary file will be used.
overwrite	if TRUE, the scheme will be overwritten if it exists
baseurl	a character string containing the base url of the repository in which the scheme definitions are located.
...	additional parameter for the function download.file.

repo	repo of the schemes.
file	if give, this file will be used as the local scheme definition, and repo will be ignored
install_package	if TRUE, install / update the accompanying R package. You can do it manually later by running scheme_install_r_package("NAME", "VERSION").
reinstall	if TRUE, the R package will be uninstalled before installing it.

Value

data.frame with two columns containing name and version of the default scheme

data.frame with two columns containing name and version of the default scheme

invisibly the value of destfile

invisibly NULL

invisibly NULL

TRUE if the theme is installed, FALSE if not

data.frame with two columns containing name and version of the intalled schemes

Returns the info about the scheme definitions in this repo as a list.

URL of the repo toi be used. If not set previously, the default repo at https://github.com/Exp-Micro-Ecol-Hub/dmdSchemeRepository/ is used.

invisibly returns the temporary location where the scheme definition is moved to.

Examples

```
scheme_active()

scheme_default()

scheme_download(
  name = "dmdScheme",
  version = "0.9.5"
)
## Not run:
scheme_install("path/to/definition.xml")
scheme_install("path/to/definition.xlsx")

## End(Not run)

## Not run:
scheme_install_r_package()

## End(Not run)

## Not run:
scheme_installed("dmdScheme", "0.9.9")
scheme_installed("dmdScheme", "0.7.3")
```

```
## End(Not run)
scheme_list_in_repo()

# returns the repo used:
scheme_repo()
## Not run:
scheme_uninstall(name = "schemename", version = "schemeversion")

## End(Not run)

scheme_list()
scheme_use(name = "dmdScheme", version = "0.9.9")
```

---

scheme_make *Functions to manage schemes*

Description

Functions to manage schemes

Usage

```
scheme_make(
  schemeDefinition,
  examples = NULL,
  install_R_package = NULL,
  path = ".",
  overwrite = FALSE,
  index_template = NULL
)
```

Arguments

schemeDefinition	path to the .xlsx file containing the **definition** of the scheme as well as the **example**
examples	character vector of directories which should be included as examples. The name of the director will be the name of the example. The example can contain a file with the name EXAMPLENAME.html where EXAMPLENAME is the name of the folder. This html will abe automatically opened when calling make_example("EXAMPLENAME") Otherwise there are no restrictions to formats.
install_R_package	path to the R script to install the R package. If NULL, no command is given.
path	where the final scheme definition should be created.
overwrite	if TRUE, the scheme definition in path will be overwritten.
index_template	the index template file which can be added

Value

fully qualified path to the created scheme

`scheme_path_index_template` *Functions to manage schemes*

Description

Functions to manage schemes

Usage

```
scheme_path_index_template()
```

Value

fully qualified path to the `index` template file

Examples

```
scheme_path_index_template()
```

`scheme_path_xlsx` *Functions to manage schemes*

Description

Functions to manage schemes

Usage

```
scheme_path_xlsx()
```

Value

fully qualified path to the `xlsx` file containing the scheme definitipn and the example, `NULL` if it does not exist.

Examples

```
scheme_path_xlsx()
```

scheme_path_xml *Functions to manage schemes*

Description

Functions to manage schemes

Usage

```
scheme_path_xml()
```

Value

fully qualified path to the `xml` file containing the scheme definitipn and the example, `NULL` if it does not exist.

Examples

```
scheme_path_xlsx()
```

toTranspose *Return tabs in scheme definition in Excel document which need to be transposed or if a tab has to be transposed*

Description

Tabs in the Excel document need to be, for processing, transposed, if they are vertical as the Experiment tab is.

Usage

```
toTranspose(tabs = NULL)
```

Arguments

`tabs`	if `NULL`, the

Value

if `tabs` is null, the `character` vector containing the sheets which need to be transposed. If `tabs` is not NULL, a `logical` vector of the same length as `tabs` which is `TRUE`, if the tab needs to be transposed, otherwise `FALSE`.

Examples

```
toTranspose()
# [1] "Experiment"     "MdBibliometric"

toTranspose(c("Experiment", "Not"))
# [1]  TRUE FALSE
```

`upgrade_old_files` *Convert older scheme versions of files to newer newer versions*

Description

Only the newest versions of `xlsx` and `xml` files can be processed by this package. To gurantee, this function provides a mechanism to convert older versions of `xlsx` and `xml` files to newer versions.

Usage

```
upgrade_old_files(file, to = scheme_active()$version)
```

Arguments

`file`	file name of `xlsx` or of `xml` file containing scheme metadata or structure
`to`	version to upgrade to. Any version supported is possible, downgrade is not supported.

Value

if a conversion has been done, file name of upgraded spreadsheet (`BASENAME(x).to.EXTENSION(x)` where `x` is the original file name and `to` is the new version), otherwise `NULL`.

Examples

```
## Not run:
upgrade("dmdScheme.xlsx")
upgrade("dmdScheme.xml")

## End(Not run)
```

valErr_extract *Extract all fields named error of class* dmdScheme_validation

Description

Extract all fields named error of class dmdScheme_validation

Usage

```
valErr_extract(x, returnRootError = FALSE)
```

Arguments

x	object of class dmdScheme_validation
returnRootError	if TRUE, return all errors **including** the error in the object x.

Value

named numeric vector of the error levels of the different validations done

valErr_info *Return info about error representation*

Description

Return info about error representation

Usage

```
valErr_info(error)
```

Arguments

error	either level, text or colour of error (see valErr_errorLevels)

Value

the row from valErr_errorLevels corresponding to the argument error

valErr_isOK *Creates* data.frame *from object of class* dmdScheme_validation *for usage in* details *of validation*

Description

Creates data.frame from object of class dmdScheme_validation for usage in details of validation

Usage

```
valErr_isOK(x, returnRootError = FALSE)
```

Arguments

x	data.frame with the fields Module, error and isOK
returnRootError	if TRUE, return all errors **including** the error in the object x.

Value

named numeric vector of the error levels of the different validations done

valErr_TextErrCol *Colour the* text *by using the error colour*

Description

Colour the text by using the error colour

Usage

```
valErr_TextErrCol(text, error, addError = TRUE)
```

Arguments

text	to be coloured. if not supplied, the coloured error text will be returned. If text is of class dmdScheme_validation, the function will be called with text = text$header, error = text$error
error	either level, text or colour of error (see valErr_errorLevels)
addError	if the error text should be added in the front of the text.

Value

the coloured text or error text

validate *Generic function to validate an object which represents a* dmdScheme

Description

This function validates an object representing a dmdScheme. The result can be used as a basis for a report by running report() on the resultiung object of class dmdScheme_validation.

Usage

```
validate(x, path = ".", validateData = TRUE, errorIfStructFalse = TRUE)

## S3 method for class 'character'
validate(x, path = ".", validateData = TRUE, errorIfStructFalse = TRUE)

## S3 method for class 'dmdSchemeSet_raw'
validate(x, path = ".", validateData = TRUE, errorIfStructFalse = TRUE)
```

Arguments

x	object referring to a dmdScheme to be validatedof class dmdSchemeSet_raw as returned from read_excel(keepData = FALSE, raw = TRUE) or file name of an xlsx file containing the metadata.
path	path to the data files
validateData	if TRUE data is validated as well; the structure is always validated
errorIfStructFalse	if TRUE an error will be raised if the schemes are not identical, i.e. there are structural differences.

Value

return the dmdScheme_validation object

Methods (by class)

- validate(character): validate a character object referring to a spreadsheet file which contains the metadata.
- validate(dmdSchemeSet_raw): validate a dmdSchemeSet_raw object

Examples

```
## validata an Excel file containing the metadata
validate(
    x = scheme_path_xlsx()
)

## validate a `dmdScheme_raw object`
```

```
validate(
   x = dmdScheme_raw()
)

## use `read_raw()` to read an Excel spreadsheet into a `dmdScheme_raw` object
x <- read_excel_raw( scheme_path_xlsx() )
validate( x = x )
```

`validateAllowedValues` *Validate allowed values*

Description

Validate allowed values

Usage

```
validateAllowedValues(sraw)
```

Arguments

`sraw`	object of type `dmdScheme_data` generated with types not converted

Value

`dmdScheme_validation` object

`validateDataFileMetaDataDataFileExists`
Validate suggested values

Description

Validate suggested values

Usage

```
validateDataFileMetaDataDataFileExists(xraw, path)
```

Arguments

`xraw`	object of type `dmdScheme_set` generated with types not converted
`path`	path to the data files

Value

`dmdScheme_validation` object

validateIDField *Validate id field*

Description

Validate id field

Usage

```
validateIDField(sraw)
```

Arguments

sraw	object of type dmdScheme_data generated with types not converted

Value

dmdScheme_validation object

validateStructure *Validate structure of* dmdScheme *object*

Description

Validate structure of dmdScheme object

Usage

```
validateStructure(x)
```

Arguments

x	object of type dmdScheme_raw

Value

dmdScheme_validation object

validateSuggestedValues *Validate suggested values*

Description

Validate suggested values

Usage

```
validateSuggestedValues(sraw)
```

Arguments

sraw	object of type dmdScheme_data generated with types not converted

Value

dmdScheme_validation object

validateTypes *Validate type of tab*

Description

Validate type of tab

Usage

```
validateTypes(sraw, sconv)
```

Arguments

sraw	object of type dmdScheme_data generated with types not converted
sconv	object of type dmdScheme_data generated with types converted

Value

dmdScheme_validation object

write_excel *Write write x as an excel file to disk*

Description

Convert the object x to an xml_document object using the function as_xml() and write it to a file. If no method as_xml() exists for the object |classx, an error will be raised.

Usage

```
write_excel(x, file, ...)
```

Arguments

x	object which can be convertewd to a dmdSchemeSet_raw odject using the function as_dmdScheme_raw which will saved as an xlsx file.
file	Path to file or connection to write to.
...	additional parameter for the conversion function (writexl::write_xlsx())

Value

invisibly returns the path to the file saved to

Examples

```
write_excel(dmdScheme(), file = tempfile())
write_excel(dmdScheme_raw(), file = tempfile())
```

write_xml *Write write x as an XML file to disk*

Description

Convert the object x to an xml_document object using the function as_xml() and write it to a file. If no method as_xml() exists for the object |classx, an error will be raised.

Usage

```
write_xml(x, file, output = "metadata", ...)
```

Arguments

x	object which will be converted to and saved as an xml file.
file	Path to file or connection to write to.
output	specifies the content and format of the exported xml. see as_xml for details
...	additional parameter for the conversion function as_xml

Examples

```
write_xml(dmdScheme(), file = tempfile())
```

Index

as_dmdScheme, 3
as_dmdScheme_raw, 4
as_eml, 5
as_xml, 6, *40*
as_xml_list, 7

cache, 8
cat_ln, 8

dmdScheme, 9
dmdScheme_example, 10, *22*
dmdScheme_raw, 10, *23*
dmdSchemeData *(*dmdScheme*)*, 9
dmdSchemeData_raw *(*dmdScheme_raw*)*, 10
dmdSchemeSet *(*dmdScheme*)*, 9
dmdSchemeSet_raw *(*dmdScheme_raw*)*, 10

format_dmdScheme_xlsx, 11

lookup_tokens, 11

make_example, 12
make_index, 13
make_new_package, 14

new_dmdScheme_validation, 15

open_new_spreadsheet, 16

pkg.dmdScheme, 17
print.dmdScheme_validation, 19
print.dmdSchemeData, 18
print.dmdSchemeSet, 18
print_dmdScheme_validation_details, 21
print_dmdScheme_validation_summary, 21

read_excel, *10*, 22
read_excel_raw, *10*, 23
read_xml, 24
report, 24
run_app, 26

scheme_active, 27
scheme_default *(*scheme_active*)*, 27
scheme_download *(*scheme_active*)*, 27
scheme_install *(*scheme_active*)*, 27
scheme_install_r_package *(*scheme_active*)*, 27
scheme_installed *(*scheme_active*)*, 27
scheme_list, *27*
scheme_list *(*scheme_active*)*, 27
scheme_list_in_repo *(*scheme_active*)*, 27
scheme_make, *15*, 30
scheme_path_index_template, 31
scheme_path_xlsx, 31
scheme_path_xml, 32
scheme_repo *(*scheme_active*)*, 27
scheme_uninstall *(*scheme_active*)*, 27
scheme_use *(*scheme_active*)*, 27

toTranspose, 32

upgrade_old_files, 33

valErr_extract, 34
valErr_info, 34
valErr_isOK, 35
valErr_TextErrCol, 35
validate, *9*, 36
validateAllowedValues, 37
validateDataFileMetaDataDataFileExists, 37
validateIDField, 38
validateStructure, 38
validateSuggestedValues, 39
validateTypes, 39

write_excel, 40
write_xml, 40