Package: datamgmt (via r-universe)

August 30, 2024

Type Package

Title Data Management Utilities for Curating, Documenting, and Publishing Data

Version 0.1.1

Description Data management utilities for curating, documenting, and publishing data.

License Apache License (== 2.0)

URL https://nceas.github.io/datamgmt/

BugReports https://github.com/NCEAS/datamgmt/issues

Encoding UTF-8

LazyData true

Imports arcticdatautils, compare, crayon, dataone, datapack, digest, dplyr, EML (>= 2.0), gsubfn, httr, jsonlite, listviewer, lubridate, magrittr, methods, ncdf4, RCurl, readxl, sf, stringi, stringr, tidyr, utils, uuid, XML, xml2, googlesheets4

Suggests testthat, knitr, igraph, purrr, rmarkdown, visNetwork

Remotes NCEAS/arcticdatautils

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

VignetteBuilder knitr

Repository https://dataoneorg.r-universe.dev

RemoteUrl https://github.com/NCEAS/datamgmt

RemoteRef HEAD

RemoteSha ac799c540cdba7ebfe82b8d161e23c7bd298cffc

Contents

`categorize_dataset` *Interim method of categorizing arctic data center datasets into one of several themes after a doi is issued*

Description

Please ask Jeanette or Jasmine to grant you access to the google sheet

Usage

```
categorize_dataset(doi, themes, coder, test = F, overwrite = F)
```

Arguments

`doi`	(character) the doi formatted as doi:10.#####/#########
`themes`	(list) themes of the dataset and you can classify up to 5. The definitions of the themes
`coder`	(character) your name, this is to identify who coded these themes
`test`	(logical) for using the test google sheet (mainly for testing purposes), defaults to FALSE
`overwrite`	(logical) whether or not to update the entry (for example if you want to update the themes)

Details

This function will account for older versions of the dataset has been already categorized. Please make sure you have a token from arcticdata.io.

Value

NULL the result will be written to an external google sheet

Author(s)

Jasmine Lai

Examples

```
## Not run:
# categorize_dataset("doi:10.18739/A2QJ77Z09", c("biology", "oceanography"), "your name")

## End(Not run)
```

`clone_object` *Clone objects between DataONE Member Nodes*

Description

Clones objects between DataONE Member Nodes. Note:

- the dateUploaded, obsoletes, and obsoletedBy fields in the sysmeta will be reset on the cloned object.
- the limit of the file size is set at a default of 1TB

Usage

```
clone_object(
  pid,
  from,
  to,
  add_access_to,
  change_auth_node,
  public = FALSE,
  new_pid = TRUE
)
```

Arguments

`pid`	(character) Object PID.
`from`	(D1Client) D1Client to clone objects from. (Objects must be public)
`to`	(D1Client) D1Client to clone objects to. (Token must be set for this node)
`add_access_to`	(character, vector) Will give read, write, and changePermission access to all strings in vector. If no additional access is desired, set to `NULL`. Note: setting this to `NULL` could lead to situations where it is not possible to read, write, or changePermission on the cloned object.

change_auth_node
: (logical) Will change the authoritativeMemberNode in the system metadata to the cloned member node if TRUE. Setting this to TRUE will allow you to edit the package after cloning. Setting this to FALSE syncs the system metadata with the package on the Authoritative Member Node, and will only allow DataONE admins with special privileges to edit the package system metadata.

public
: (logical) Optional. Will set public read access. Defaults to FALSE.

new_pid
: (logical) Optional. Will give the clone a new PID. Defaults to TRUE.

Value

(character) PID of cloned object. NULL if could not clone.

Examples

```
## Not run:
# First set up the member nodes we're cloning between
# (in this example they are the same but could be different)
to <- dataone::D1Client("STAGING", "urn:node:mnTestARCTIC")
from <- dataone::D1Client("STAGING", "urn:node:mnTestARCTIC")

# Choose an object to clone (here a new one is created)
pid <- arcticdatautils::create_dummy_object(to@mn)

# Clone object
cloned_pid <- clone_object(pid = pid,
                           from = from,
                            to = to,
                            add_access_to = arcticdatautils:::get_token_subject(),
                            change_auth_node = TRUE,
                            public = TRUE,
                            new_pid = TRUE)

## End(Not run)
```

---

clone_package *Clone packages between DataONE Member Nodes*

Description

This function copies a data package from one DataONE Member Node to another. Note: the dateUploaded, obsoletes, and obsoletedBy fields in the sysmeta will be reset on the cloned object. This will not update the information in the metadata object. This can also be used to restore an older version of a package to a Member Node, provided that the user subsequently obsoletes the version of the package that they used to create the clone.

Usage

```
clone_package(
  resource_map_pid,
  from,
  to,
  add_access_to,
  change_auth_node,
  public = FALSE,
  clone_children = FALSE,
  new_pid = TRUE
)
```

Arguments

resource_map_pid	(character) PID for the package resource map.
from	(D1Client) D1Client to clone package from. (Package must be public)
to	(D1Client) D1Client to clone package to. (Token must be set for this node)
add_access_to	(character) Will give read, write, and changePermission access to all strings in vector. If no additional access is desired, set to NULL. Note: setting this to NULL could lead to situations where it is not possible to read, write, or changePermissions on the cloned object.
change_auth_node	(logical) Will change the authoritativeMemberNode in the system metadata to the cloned member node if TRUE. Setting this to TRUE will allow you to edit the package after cloning. Setting this to FALSE syncs the system metadata with the package on the Authoritative Member Node, and will only allow DataONE admins with special privileges to edit package system metadata.
public	(logical) Optional. Will set public read access. Defaults to FALSE.
clone_children	(logical) Optional. Will clone all children recursively if TRUE. Defaults to FALSE.
new_pid	(logical) Optional. Will give the clone a new PID. Defaults to TRUE.

Author(s)

Dominic Mullen, <dmullen17@gmail.com>

See Also

copy_package()

Examples

```
## Not run:
# First set up the member nodes we're cloning between
# (in this example they are the same but could be different)
to <- dataone::D1Client("STAGING", "urn:node:mnTestARCTIC")
from <- dataone::D1Client("STAGING", "urn:node:mnTestARCTIC")
```

```
# Choose a package to clone (here a new one is created)
package <- arcticdatautils::create_dummy_package(to@mn)

# Clone object
cloned_package <- clone_package(resource_map_pid = package$resource_map
                          from = from,
                           to = to,
                           add_access_to = arcticdatautils:::get_token_subject(),
                           change_auth_node = TRUE,
                           public = TRUE,
                           new_pid = TRUE)

## End(Not run)
```

---

copy_package *Copy packages between DataONE Member Nodes with new identifiers*

Description

This function is a convenience wrapper around `clone_package()` that copies a package rather than cloning it. The distinction is that new PIDs will always be generated, and the system metadata will reflect a stand-alone package rather than a clone. This function copies a data package from one DataONE Member Node to another, with new identifiers This can also be used to restore an older version of a package to a Member Node, provided that the user subsequently obsoletes the version of the package that they used to create the copy using `obsolete_package()`.

Usage

```
copy_package(
  resource_map_pid,
  from,
  to,
  public = FALSE,
  clone_children = FALSE
)
```

Arguments

`resource_map_pid`	(character) Object pid
`from`	(D1Client) D1Client to clone package from. (Token must be set for this node)
`to`	(D1Client) D1Client to clone package to. (Token must be set for this node)
`public`	(logical) Optional. Will set public read access. Defaults to `FALSE`.
`clone_children`	(logical) Optional. Will clone all children recursively if TRUE. Defaults to `FALSE`.

Author(s)

Dominic Mullen, <dmullen17@gmail.com>

See Also

clone_package() obsolete_package()

Examples

```
## Not run:
# First set up the member nodes we're copying between
# (in this example they are the same but could be different)
to <- dataone::D1Client("STAGING", "urn:node:mnTestARCTIC")
from <- dataone::D1Client("STAGING", "urn:node:mnTestARCTIC")

# Choose a package to copy (here a new one is created)
package <- arcticdatautils::create_dummy_package(to@mn)

copied_package <- clone_package(resource_map_pid = package$resource_map,
                                from = from,
                                to = to)

## End(Not run)
```

---

datamgmt *Data management utilities for curating, documenting, and publishing data*

Description

The datamgmt R package supports management of data packages on the Arctic Data Center (ADC) and State of Alaska's Salmon and People (SASAP) data portals.

Author(s)

NCEAS Data Science Fellows

See Also

datamgmt pkgdown website

`download_eml_attributes` *Download attribute (column) metadata from a DataONE metadata object to csv files*

Description

Download attribute metadata from an EML object as csvs. The name of each csv corresponds to the file name of the Data Object it describes. This can be prepended with the package identifier by setting `prefix_file_names = TRUE` (recommended).

Usage

```
download_eml_attributes(doc, download_directory, prefix_file_names = FALSE)
```

Arguments

`doc`	(emld) EML object.
`download_directory`	(character) Directory to download attribute metadata csvs to.
`prefix_file_names`	(logical) Optional. Whether to prefix file names with the package metadata identifier. This is useful when downloading files from multiple packages to one directory.

Author(s)

Dominic Mullen, <dmullen17@gmail.com>

Examples

```
## Not run:
mn <- dataone::getMNode(cn, 'urn:node:ARCTIC')
doc <- EML::read_eml(rawToChar(dataone::getObject(mn, "doi:10.18739/A23W02")))
attributes <- datamgmt::download_eml_attributes(doc, download_directory = tempdir(),
prefix_file_names = TRUE)

## End(Not run)
```

`download_package` *Download a data package (optionally with child packages)*

Description

This function downloads all of the data objects in a data package to the local filesystem. It is particularly useful when a data package is too large to download using the web interface.

Usage

```
download_package(
  mn,
  resource_map_pid,
  download_directory,
  prefix_file_names = TRUE,
  download_column_metadata = FALSE,
  convert_excel_to_csv = FALSE,
  download_child_packages = TRUE,
  check_download_size = FALSE
)
```

Arguments

`mn`	(MNode) The Member Node to download from.
`resource_map_pid`	(chraracter) The PID of the resource map for the package to download.
`download_directory`	(character) The path of the directory to download the package to.
`prefix_file_names`	(logical) Optional. Whether to prefix file names with the package metadata identifier. This is useful when downloading files from multiple packages to one directory.
`download_column_metadata`	(logical) Optional. Whether to download attribute (column) metadata as csv files. If using this, then it is recommended to also set `prefix_file_names = TRUE`.
`convert_excel_to_csv`	(logical) Optional. Whether to convert Excel files to csv files. The csv files are downloaded as sheetName_excelWorkbookName.csv
`download_child_packages`	(logical) Optional. Whether to download data from child packages of the selected package. Defaults to `TRUE`.
`check_download_size`	(logical) Optional. Whether to check the total download size before continuing. Setting this to `FALSE` speeds up the function, especially when the package has many elements.

Details

Setting check_download_size to TRUE is recommended if you are uncertain of the total download size and want to avoid downloading very large data packages.

This function will also download any data objects it finds in any child data packages of the input data package. If you would only like to download data from one data package, set download_child_packages to FALSE.

Author(s)

Dominic Mullen, <dmullen17@gmail.com>

Examples

```
## Not run:
cn <- CNode("PROD")
mn <- getMNode(cn, "urn:node:ARCTIC")

download_package(mn, "resource_map_urn:uuid:2b4e4174-4e4b-4a46-8ab0-cc032eda8269",
"/home/dmullen")

## End(Not run)
```

---

download_packages *Download multiple data packages*

Description

This function is a convenience wrapper for download_package() when downloading multiple data packages. It downloads all of the data objects in a data package to the local filesystem. It is particularly useful when a data package is too large to download using the web interface.

Usage

```
download_packages(mn, resource_map_pids, download_directory, ...)
```

Arguments

mn	(MNode) The Member Node to download from.
resource_map_pids	(chraracter) The PIDs of the resource maps for the packages to download.
download_directory	(character) The path of the directory to download the packages to.
...	Allows arguments from download_package().

Details

Setting check_download_size to TRUE is recommended if you are uncertain of the total download size and want to avoid downloading very large data packages.

This function will also download any data objects it finds in any child data packages of the input data package. If you would only like to download data from one data package, set download_child_packages to FALSE.

Author(s)

Dominic Mullen, <dmullen17@gmail.com>

See Also

download_package()

Examples

```
## Not run:
cn <- CNode("PROD")
mn <- getMNode(cn, "urn:node:ARCTIC")

download_packages(mn, c("resource_map_doi:10.18739/A21G1P", "resource_map_doi:10.18739/A2RZ6X"),
"/home/dmullen/downloads", prefix_file_names = TRUE, download_column_metadata = TRUE,
convert_excel_to_csv = TRUE)

## End(Not run)
```

---

edit_attribute *Edit a single attribute*

Description

This function edits the slots of a single attribute in an existing list of attributes for a data object.

Usage

```
edit_attribute(
  attribute,
  attributeName = NULL,
  attributeLabel = NULL,
  attributeDefinition = NULL,
  domain = NULL,
  measurementScale = NULL,
  unit = NULL,
  numberType = NULL,
  definition = NULL,
  formatString = NULL,
```

```
  missingValueCode = NULL,
  missingValueCodeExplanation = NULL
)
```

Arguments

attribute	(attribute) The attribute in the the attributeList of a data object.
attributeName	(character) The new name to give to the attribute.
attributeLabel	(character) The new label to give to the attribute.
attributeDefinition	(character) The new attributeDefinition to give to the attribute.
domain	(character) The new domain to give to the attribute.
measurementScale	(character) The new measurementScale to give to the attribute.
unit	(character) The new unit (for numericDomain) to give to the attribute.
numberType	(character) The new numberType (for numericDomain) to give to the attribute.
definition	(character) The new definition (for textDomain) to give to the attribute.
formatString	(character) The new formatString (for dateTime) to give to the attribute.
missingValueCode	(character) The new missing value code to give to the attribute.
missingValueCodeExplanation	(character) The new missing value code explanation to give to the attribute.

Details

This function can only be used on attributes entirely defined within the 'attributes' slot of attributeList; it cannot be used to edit the factors table of an enumeratedDomain.

In cases with very large attribute lists, use `arcticdatautils::which_in_eml()` first to locate the attribute index number in the attributeList.

Value

(attribute) The modified attribute.

Examples

```
## Not run:
cn <- dataone::CNode('PROD')
mn <- dataone::getMNode(cn, 'urn:node:ARCTIC')
doc <- EML::read_eml(rawToChar(dataone::getObject(mn, "doi:10.18739/A23W02")))
new_attribute <- edit_attribute(doc$dataset$dataTable[[1]]$attributeList$attribute[[1]],
                                attributeName = "new name")
doc$dataset$dataTable[[1]]$attributeList$attribute[[1]] <- new_attribute

# Change an attribute's measurementScale from ratio to nominal
# (requires updating domain to textDomain or enumeratedDomain, setting unit and numberType
# to NA and adding a setting definition
```

```
new_attribute <- edit_attribute(doc$dataset$dataTable[[1]]$attributeList$attribute[[1]],
                          domain = "textDomain", measurementScale = "nominal", unit = NA,
                               numberType = NA, definition = 'new definition')
doc$dataset$dataTable[[1]]$attributeList$attribute[[1]] <- new_attribute
EML::eml_validate(doc) # validating complex EML changes is usually a good idea

# Add the same missing value codes to all attributes for a data object
new_attributes <- lapply(doc$dataset$dataTable[[1]]$attributeList$attribute, edit_attribute,
                 missingValueCode = "NA", missingValueCodeExplanation = "data unavailable")
doc$dataset$dataTable[[1]]$attributeList$attribute <- new_attributes

## End(Not run)
```

`excel_to_csv` *Convert Excel workbook to multiple csv files*

Description

Converts an Excel workbook into multiple csv files (one per tab). Names the files in the following format: sheetName_excelName.csv.

Usage

```
excel_to_csv(path, directory = NULL, ...)
```

Arguments

`path`	(character) File location of the excel workbook.
`directory`	(character) Optional. Directory to download csv files to.
`...`	Optional. Allows arguments from read_excel. Defaults to the base directory that `path` is located in.

Value

invisible

Author(s)

Dominic Mullen <dmullen17@gmail.com>

`get_awards` *Get NSF Arctic/Polar program award information*

Description

Uses the NSF API to get all records pertaining to the Arctic or Polar programs.

Usage

```
get_awards(from_date = NULL, to_date = NULL, query = NULL, print_fields = NULL)
```

Arguments

`from_date`	(character) Optional. Returns all records with start date after specified date. Format = *mm/dd/yyyy*
`to_date`	(character) Optional. Returns all records with start date before specified date. Format = *mm/dd/yyyy*
`query`	(character) Optional. By default, the function searches for all awards with either "polar" or "arctic" in the fundProgramName. Additional queries can be specified as defined in the NSF API. Use '&' to join multiple queries (i.e., *keyword=water&agency=NASA*)
`print_fields`	(character) Optional. By default, the following fields will be returned: id, date, startDate, expDate, fundProgramName, poName, title, awardee, piFirstName, piLastName, piPhone, piEmail. Additional field names can be found in the printFields description of the NSF API.

Author(s)

Irene Steves

Examples

```
## Not run:
all_awards <- get_awards()
new_awards <- get_awards(from_date = "01/01/2017")

## End(Not run)
```

get_eml_attributes *Return attribute (column) metadata from a DataONE metadata object*

Description

Return attribute metadata from an EML object. This is largely a wrapper for the function EML::get_attributes().

Usage

```
get_eml_attributes(doc)
```

Arguments

doc	(emld) EML object.

Value

(list) A list of all attribute metadata from the EML in data.frame objects

Author(s)

Dominic Mullen, <dmullen17@gmail.com>

Examples

```
## Not run:
cn <- dataone::CNode('PROD')
mn <- dataone::getMNode(cn, 'urn:node:ARCTIC')
doc <- EML::read_eml(rawToChar(dataone::getObject(mn, "doi:10.18739/A23W02")))
attributes <- get_eml_attributes(doc)

# switch nodes
cn <- dataone::CNode('PROD')
knb <- dataone::getMNode(cn,"urn:node:KNB")
doc <- EML::read_eml(rawToChar(dataone::getObject(knb, "doi:10.5063/F1639MWV")))
attributes <- get_eml_attributes(doc)

## End(Not run)
```

get_eml_attributes_url *Return attribute (column) metadata from a DataONE package URL*

Description

Return attribute metadata from an EML object or DataONE package URL. This is largely a wrapper for the function `EML::get_attributes()`.

Usage

```
get_eml_attributes_url(
  mn,
  url_path,
  write_to_csv = FALSE,
  prefix_file_names = FALSE,
  download_directory = NULL
)
```

Arguments

`mn`	(MNode/CNode) The DataONE Node that stores the Metadata object, from `https://cn.dataone.org/cn/v2/node`
`url_path`	(character) The URL of the DataONE Package.
`write_to_csv`	(logical) Optional. Option whether to download the attribute metadata to csv files. Defaults to `FALSE`
`prefix_file_names`	(logical) Optional. Whether to prefix file names with the package metadata identifier. This is useful when downloading files from multiple packages to one directory.
`download_directory`	(character) Optional. Directory to download attribute metadata csv files to. Required if `write_to_csv` is `TRUE`.

Value

(list) A list of all attribute metadata from the EML in data.frame objects.

Author(s)

Dominic Mullen, `<dmullen17@gmail.com>`

Examples

```
## Not run:
attributes <- get_eml_attributes(mn,
"https://arcticdata.io/catalog/#view/doi:10.18739/A23W02")

# Download attribute metadata in csv format:
attributes <- get_eml_attributes(mn,
"https://arcticdata.io/catalog/#view/doi:10.18739/A23W02",
write_to_csv = TRUE,
download_directory = tempdir())
# switch nodes
cn <- dataone::CNode('PROD')
knb <- dataone::getMNode(cn,"urn:node:KNB")
attributes <- get_eml_attributes(knb,
"https://knb.ecoinformatics.org/#view/doi:10.5063/F1639MWV")

## End(Not run)
```

---

guess_member_node *Guess the Member Node*

Description

Guess the Member Node that stores a DataONE object based on its unique identifier (pid) and coordinating node (cn). In most cases the object is stored on the Production ("PROD") Node, however this function can search across all coordinating nodes. If only one Member Node is identified this function returns the Member Node as a DataONE "MNode" object. If multiple Member Nodes are identified a vector of nodes is printed.

Usage

```
guess_member_node(pid, cn = "PROD")
```

Arguments

pid	(character) The DataONE unique object identifier. A DataONE package URL can also be used as input (although this method is less reliable).
cn	(character) A character vector of coordinate nodes to search. Defaults to "PROD". Can be set to any combination of ("PROD", "STAGING", "STAGING2", "SANDBOX", "SANDBOX2", "DEV", "DEV2").

Author(s)

Dominic Mullen, <dmullen17@gmail.com>

Examples

```
## Not run:
The following two calls are equivalent:
mn <- guess_member_node("doi:10.18739/A2G287")
mn <- guess_member_node("doi:10.18739/A2G287", "PROD")

Use a DataONE package URL
mn <- guess_member_node("https://arcticdata.io/catalog/#view/doi:10.18739/A2TX35587")
Search all coordinating nodes:
cn <- c("PROD", "STAGING", "STAGING2", "SANDBOX", "SANDBOX2", "DEV", "DEV2")
mn <- guess_member_node("doi:10.18739/A2G287", cn)

## End(Not run)
```

`obsolete_package` *Obsolete a DataONE Package with a new version*

Description

This function obsoletes a DataONE package with a newer version by merging the two version chains. The ideal use case for this function is when the only option to fix a broken package is by re-uploading a previous version and merging the two version chains. In other cases `arcticdatautils::publish_update()` should be used.

Usage

```
obsolete_package(mn, metadata_obsolete, metadata_new)
```

Arguments

`mn`	(MNode) The DataONE member node.
`metadata_obsolete`	(character) The metadata PID of the old, or broken, version. Any metadata PID from the obsolete version chain can be used - sets the PID to the end of the version chain.
`metadata_new`	(character) The metadata PID of the new version. Any metadata PID from the new version chain can be used - sets the PID to the beginning of the version chain.

Value

(logical) `TRUE/FALSE`

Author(s)

Dominic Mullen, `<dmullen17@gmail.com>`

Examples

```
## Not run:
cn <- dataone::CNode("STAGING")
mn <- dataone::getMNode(cn,"urn:node:mnTestARCTIC")

pkg_old <- arcticdatautils::create_dummy_package(mn)
pkg_new <- arcticdatautils::create_dummy_package(mn)

obsolete_package(mn, pkg_old$metadata, pkg_new$metadata)

## End(Not run)
```

---

plot_pkg_structure *Plot package structure*

Description

This function visualizes how data packages in a data package family are related to each other.

Usage

```
plot_pkg_structure(mn, parent_rm_pid)
```

Arguments

mn	(MNode) The Member Node to query.
parent_rm_pid	(character) The top-level PID in a data package family.

Value

A visIgraph plot.

Examples

```
## Not run:
cn <- CNode("PROD")
mn <- getMNode(cn,"urn:node:ARCTIC")

parent_rm_pid <- "resource_map_urn:uuid:..."

plot_pkg_structure(mn, parent_rm_pid)

## End(Not run)
```

`qa_attributes` *Check congruence of data and metadata attributes for a tabular data object*

Description

This function checks the congruence of data and metadata attributes for a tabular data object. Supported objects include `dataTable`, `otherEntity`, and `spatialVector` entities. It can be used on its own but is also called by `qa_package()` to check all tabular data objects in a data package.

Usage

```
qa_attributes(entity, data, doc = NULL)
```

Arguments

`entity`	(emld) An EML `dataTable`, `otherEntity`, or `spatialVector` associated with the data object.
`data`	(data.frame) A data frame of the data object.
`doc`	(emld) The entire EML object. This is necessary if attributes with references are being checked.

Details

This function checks the following:

- Names: Check that column names in attributes match column names in data frame. Possible conditions to check for:
  - attributeList does not exist for data frame
  - Some of the attributes that exist in the data do not exist in the attributeList
  - Some of the attributes that exist in the attributeList do not exist in the data
  - Typos in attribute or column names resulting in nonmatches
- Domains: Check that attribute types in EML match attribute types in data frame. Possible conditions to check for:
  - nominal, ordinal, integer, ratio, dateTime
  - If domain is enumerated domain, enumerated values in the data are accounted for in the enumerated definition
  - If domain is enumerated domain, enumerated values in the enumerated definition are all represented in the data
  - Type of data does not match attribute type
- Values: Check that values in data are reasonable. Possible conditions to check for:
  - Accidental characters in the data (e.g., one character in a column of integers)
  - If missing values are present, missing value codes are also present

Value

NULL

See Also

qa_package()

Examples

```
## Not run:
# Checking a .csv file
dataTable <- doc$dataset$dataTable[[1]]
data <- readr::read_csv("https://cn.dataone.org/cn/v2/resolve/urn:uuid:...")

qa_attributes(dataTable, data)

## End(Not run)
```

---

qa_package *Check package including congruence of attributes and data*

Description

This function checks that the attributes listed in the metadata match the values in the data for each tabular data object. It may also optionally check if all creators have ORCIDs and have full access to all elements of the data package.

Usage

```
qa_package(
  mn,
  resource_map_pid,
  read_all_data = TRUE,
  check_attributes = TRUE,
  check_creators = FALSE,
  check_access = FALSE
)
```

Arguments

mn	(MNode) The Member Node to query.
resource_map_pid	(character) The PID for a resource map.
read_all_data	(logical) Read all data from remote and check that column types match attributes. If FALSE, only read first 10 rows. Only applicable to public packages (private packages will read complete dataset). If check_attributes = FALSE, no rows will be read.

check_attributes
: (logical) Check congruence of attributes and data.

check_creators
: (logical) Check if each creator has an ORCID. Will also run if check_access = TRUE.

check_access
: (logical) Check if each creator has full access to the metadata, resource map, and data objects. Will not run if the checks associated with check_creators fail.

Value

NULL

Examples

```
## Not run:
# Run all QA checks

qa_package(mn, pid, read_all_data = TRUE, check_attributes = TRUE,
           check_creators = TRUE, check_access = TRUE)

## End(Not run)
```

---

query_all_versions *Solr query all versions of a PID*

Description

This function uses a combination of arcticdatautils::get_all_versions() and a Solr query to return the query fields for all versions of the specified PID. Each row of the resulting data frame corresponds to a version and the columns are the query fields.

Usage

```
query_all_versions(mn, object_pid, fields = "*")
```

Arguments

mn
: (MNode) The Member Node where the object should be searched for.

object_pid
: (character) PID for the object that you want to return information about.

fields
: (character) List of fields that you want returned in the data frame. Default returns all non NULL fields.

Value

(data.frame) Data frame with rows for each version of the PID and columns with each query field.

Author(s)

Sharis Ochs, <sharisnochs@gmail.com>

Examples

```
## Not run:
cn <- dataone::CNode("PROD")
mn <- dataone::getMNode(cn, "urn:node:ARCTIC")

df <- query_all_versions(mn, "doi:10.18739/A27D2Q670", c("id", "title", "origin", "submitter"))
View(df)

## End(Not run)
```

Index