Package: datapack (via r-universe)

October 29, 2026

Title A Flexible Container to Transport and Manipulate Data and Associated Resources

Version 1.4.2

Description Provides a flexible container to transport and manipulate complex sets of data. These data may consist of multiple data files and associated meta data and ancillary files. Individual data objects have associated system level meta data, and data files are linked together using the OAI-ORE standard resource map which describes the relationships between the files. The OAI- ORE standard is described at <https://www.openarchives.org/ore/>. Data packages can be serialized and transported as structured files that have been created following the BagIt specification. The BagIt specification is described at <https://datatracker.ietf.org/doc/html/draft-kunze-bagit-08>.

Date 2025-10-07

Depends R (>= 3.1.1)

Imports digest, fs, methods, redland, XML, utils, uuid, zip

Suggests testthat (>= 3.0.0), knitr, httr, igraph, rmarkdown

License Apache License (== 2.0)

URL https://docs.ropensci.org/datapack/, https://github.com/ropensci/datapack

BugReports https://github.com/ropensci/datapack/issues

VignetteBuilder knitr

Encoding UTF-8

Language en-US

Collate 'Constants.R' 'SystemMetadata.R' 'dmsg.R' 'DataObject.R' 'DataPackage.R' 'ResourceMap.R' 'datapack-package.r' 'zzz.R'

RoxygenNote 7.3.3

Config/testthat/edition 3

Config/pak/sysreqs cmake make libuv1-dev libxml2-dev librdf0-dev

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

Date/Publication 2025-10-10 00:00:06 UTC

RemoteUrl https://github.com/ropensci/datapack

RemoteRef HEAD

RemoteSha c865aa532fa08c63098f2771ca80ba768f3c743c

Contents

addAccessRule . . . 3
addData . . . 5
addMember . . . 6
calculateChecksum . . . 7
canRead . . . 8
clearAccessPolicy . . . 9
containsId . . . 10
createFromTriples . . . 11
DataObject-class . . . 12
DataPackage-class . . . 14
describeWorkflow . . . 16
dmsg . . . 17
freeResourceMap . . . 18
getData . . . 18
getFormatId . . . 19
getIdentifier . . . 20
getIdentifiers . . . 21
getMember . . . 21
getRelationships . . . 22
getSize . . . 23
getTriples . . . 24
getValue . . . 25
hasAccessRule . . . 26
initialize,DataObject-method . . . 27
initialize,DataPackage-method . . . 29
initialize,ResourceMap-method . . . 30
initialize,SystemMetadata-method . . . 30
insertRelationship . . . 33
parseRDF . . . 34
parseSystemMetadata . . . 35
plotRelationships . . . 36
recordDerivation . . . 37
removeAccessRule . . . 38
removeMember . . . 40
removeRelationships . . . 41
replaceMember . . . 42
ResourceMap-class . . . 44

selectMember 45
serializePackage 46
serializeRDF 48
serializeSystemMetadata 49
serializeToBagIt 50
setPublicAccess 52
setValue 53
SystemMetadata 54
SystemMetadata-class 55
updateMetadata 56
updateRelationships 58
updateXML 58
validate 59

Index **61**

`addAccessRule` *Add access rules to the specified object*

Description

Add one or more access rules to the access policy of the specified object.

Usage

```
addAccessRule(x, ...)

## S4 method for signature 'SystemMetadata'
addAccessRule(x, y, ...)

## S4 method for signature 'DataObject'
addAccessRule(x, y, ...)

## S4 method for signature 'DataPackage'
addAccessRule(x, y, ...)
```

Arguments

`x`	The object instance to which to add the rules
`...`	Additional arguments • `permission`: The permission to be applied to subject if x is character (read, write, changePermission)
`y`	The subject of the rule to be added, or a data frame of subject/permission tuples

Details

If the y argument is specified as a character string containing a `subject`, then an optional `permission` parameter must be specified, that contains a character list specifying the permissions to add for each `subject`.

Note that when `addAccessRule` is called with a 'DataPackage' argument, the additional parameter `identifiers` can be used:

- `identifiers`: A list of `character` values containing package member identifiers that the access rule will be applied to (all members is the default).

Value

The SystemMetadata object with the updated access policy.

The DataObject with the updated access policy

The DataPackage with updated DataObject access policies

See Also

SystemMetadata-class

DataObject-class

DataPackage-class

Examples

```
# Add an access rule to a SystemMetadata access policy.
# Parameter "y" can be character string containing the subject of the access rule:
sysmeta <- new("SystemMetadata")
sysmeta <- addAccessRule(sysmeta, "uid=smith,ou=Account,dc=example,dc=com", "write")
accessRules <- data.frame(subject=c("uid=smith,ou=Account,dc=example,dc=com",
 "uid=slaughter,o=unaffiliated,dc=example,dc=org"), permission=c("write", "changePermission"))
sysmeta <- addAccessRule(sysmeta, accessRules)
# Alternatively, parameter "y" can be a data.frame containing one or more access rules:
sysmeta <- addAccessRule(sysmeta, "uid=smith,ou=Account,dc=example,dc=com", "write")
accessRules <- data.frame(subject=c("uid=smith,ou=Account,dc=example,dc=com",
 "uid=slaughter,o=unaffiliated,dc=example,dc=org"), permission=c("write", "changePermission"))
sysmeta <- addAccessRule(sysmeta, accessRules)
# Add an access rule to a DataObject
data <- charToRaw("1,2,3\n4,5,6\n")
obj <- new("DataObject", id="1234", dataobj=data, format="text/csv")
obj <- addAccessRule(obj, "uid=smith,ou=Account,dc=example,dc=com", "write")
# Add an access rule to members of a DataPackage
# First create a sample DataPackage
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6\n")
obj <- new("DataObject", id="id1", dataobj=data, format="text/csv")
dp <- addMember(dp, obj)
data2 <- charToRaw("7,8,9\n4,10,11\n")
obj2 <- new("DataObject", id="id2", dataobj=data2, format="text/csv")
dp <- addMember(dp, obj2)
```

```
# Add access rule to all package members
dp <- addAccessRule(dp, "uid=smith,ou=Account,dc=example,dc=com", "write", getIdentifiers(dp))
```

---

addData *Add a DataObject to the DataPackage*

Description

The DataObject is added to the DataPackage.

Usage

```
addData(x, do, ...)

## S4 method for signature 'DataPackage,DataObject'
addData(x, do, mo = NA_character_)
```

Arguments

x	A DataPackage instance
do	A DataObject instance
...	(Additional parameters)
mo	A DataObject (containing metadata describing "do") to associate with the science object.

Details

The DataObject "do" is added to the DataPackage. If the optional "mo" parameter is specified, then it is assumed that the DataObject "mo" is a metadata object that describes the science object "do" that is being added. The addData function will add a relationship to the DataPackage resource map that indicates that the metadata object describes the science object using the Citation Typing Ontology (CITO). Note: this method updates the passed-in DataPackage object. documents and isDocumentedBy relationship.

Value

the updated DataPackage object

See Also

DataPackage-class

Examples

```
dpkg <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
metadata <- charToRaw("EML or other metadata document text goes here\n")
md <- new("DataObject", id="md1", dataobj=metadata, format="text/xml", user="smith",
  mnNodeId="urn:node:KNB")
do <- new("DataObject", id="id1", dataobj=data, format="text/csv", user="smith",
  mnNodeId="urn:node:KNB")
# Associate the metadata object with the science object. The 'mo' object will be added
# to the package  automatically, since it hasn't been added yet.
# This method is now deprecated, so suppress warnings if desired.
suppressWarnings(dpkg <- addData(dpkg, do, md))
```

`addMember` *Add a DataObject to the DataPackage*

Description

The DataObject is added to the DataPackage.

Usage

```
addMember(x, ...)

## S4 method for signature 'DataPackage'
addMember(x, do, mo = NA_character_)
```

Arguments

`x`	A DataPackage instance
`...`	(Additional parameters)
`do`	The DataObject to add.
`mo`	A DataObject (containing metadata describing `"do"`) to associate with the science object. If this DataObject has already been added to the package, the argument can be a `"character"` containing the DataObject identifier.

Details

The DataObject `"do"` is added to the DataPackage. If the optional `"mo"` parameter is specified, then it is assumed that the DataObject `"mo"` is a metadata object that describes the science object `"do"` that is being added. The `addMember` function will add a relationship to the DataPackage resource map that indicates that the metadata object describes the science object using the Citation Typing Ontology (CITO). Note: this method updates the passed-in DataPackage object. `documents` and `isDocumentedBy` relationship.

Value

the updated DataPackage object

See Also

DataPackage-class

Examples

```
dpkg <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
metadata <- charToRaw("EML or other metadata document text goes here\n")
md <- new("DataObject", id="md1", dataobj=metadata, format="text/xml", user="smith",
  mnNodeId="urn:node:KNB")
do <- new("DataObject", id="id1", dataobj=data, format="text/csv", user="smith",
  mnNodeId="urn:node:KNB")
# Associate the metadata object with the science object. The 'mo' object will be added
# to the package  automatically, since it hasn't been added yet.
dpkg <- addMember(dpkg, do, md)
```

---

calculateChecksum *Calculate a checksum for the DataObject using the specified checksum algorithm*

Description

calculates a checksum

Usage

```
calculateChecksum(x, ...)

## S4 method for signature 'DataObject'
calculateChecksum(x, checksumAlgorithm = "SHA256", ...)
```

Arguments

x	A DataObject instance
...	Additional parameters (not yet used)
checksumAlgorithm	a character value specifying the checksum algorithm to use (i.e "MD5" or "SHA1" or "SHA256")

Value

The calculated checksum

Note

this method is intended for internal package use only.

canRead *Test whether the provided subject can read an object*

Description

Using the AccessPolicy, tests whether the subject has read permission for the object. This method is meant work prior to submission to a repository, and will show the permissions that would be enforced by the repository on submission. Currently it only uses the AccessPolicy to determine who can read (and not the rightsHolder field, which always can read an object). If an object has been granted read access by the special "public" subject, then all subjects have read access.

Usage

```
canRead(x, ...)

## S4 method for signature 'DataObject'
canRead(x, subject)
```

Arguments

x	DataObject
...	Additional arguments
subject	: the subject name of the person/system to check for read permissions

Details

The subject name used in both the AccessPolicy and in the 'subject' argument to this method is a string value, but is generally formatted as an X.509 name formatted according to RFC 2253.

Value

boolean TRUE if the subject has read permission, or FALSE otherwise

See Also

DataObject-class

Examples

```
data <- charToRaw("1,2,3\n4,5,6\n")
obj <- new("DataObject", id="1234", dataobj=data, format="text/csv")
obj <- addAccessRule(obj, "smith", "read")
access <- canRead(obj, "smith")
```

clearAccessPolicy *Clear the accessPolicy from the specified object*

Description

Clears the accessPolicy from the specified object by overwriting all existing access rules set on the object with an empty set.

Usage

```
clearAccessPolicy(x, ...)

## S4 method for signature 'SystemMetadata'
clearAccessPolicy(x, ...)

## S4 method for signature 'DataObject'
clearAccessPolicy(x, ...)

## S4 method for signature 'DataPackage'
clearAccessPolicy(x, identifiers = list(), ...)
```

Arguments

x	the instance to clear access rules from.
...	(Additional parameters)
identifiers	A list of character values containing package member identifiers that the access rule will be applied to.

Value

The SystemMetadata object with the cleared access policy.

The DataObject with the cleared access policy.

The SystemMetadata object with the cleared access policy.

See Also

SystemMetadata-class

DataObject-class

DataPackage-class

Examples

```
# Clear access policy for a SystemMetadata object.
sysmeta <- new("SystemMetadata")
sysmeta <- addAccessRule(sysmeta, "uid=smith,ou=Account,dc=example,dc=com", "write")
sysmeta <- clearAccessPolicy(sysmeta)
# Clear access policy for a DataObject
do <- new("DataObject", format="text/csv", filename=system.file("extdata/sample-data.csv",
          package="datapack"))
do <- addAccessRule(do, "uid=smith,ou=Account,dc=example,dc=com", "write")
do <- clearAccessPolicy(do)
# Clear access policy for a DataPackage
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6\n")
obj <- new("DataObject", dataobj=data, format="text/csv")
dp <- addMember(dp, obj)
data2 <- charToRaw("7,8,9\n4,10,11\n")
obj2 <- new("DataObject", dataobj=data2, format="text/csv")
dp <- addMember(dp, obj2)

# Add the access rule to all package members
dp <- addAccessRule(dp, "uid=smith,ou=Account,dc=example,dc=com",
    permission="write")
# Now clear the access policy for just the second object
dp <- clearAccessPolicy(dp, getIdentifier(obj2))
```

---

containsId *Returns true if the specified object is a member of the package*

Description

Returns true if the specified object is a member of the package

Usage

```
containsId(x, ...)

## S4 method for signature 'DataPackage'
containsId(x, identifier)
```

Arguments

x	A DataPackage object
...	(Not yet used)
identifier	The DataObject identifier to check for inclusion in the DataPackage

Value

A logical - a value of TRUE indicates that the DataObject is in the DataPackage

See Also

DataPackage-class

Examples

```
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
id <- "myNewId"
do <- new("DataObject", id=id, dataobj=data, format="text/csv", user="jsmith")
dp <- addMember(dp, do)
isInPackage <- containsId(dp, identifier="myNewId")
```

---

createFromTriples *Populate a ResourceMap with RDF relationships from data.frame*

Description

RDF relationships are added to a ResourceMap object from a data.frame that contains RDF triples. For example, relationships can be exported from a DataPackage via getRelationships. The resulting data.frame is then read by createFromTriples to create the ResourceMap.

Usage

```
createFromTriples(x, ...)

## S4 method for signature 'ResourceMap'
createFromTriples(
  x,
  relations,
  identifiers,
  resolveURI = NA_character_,
  externalIdentifiers = list(),
  creator = NA_character_,
  ...
)
```

Arguments

x	a ResourceMap
...	(Additional parameters)
relations	A data.frame to read relationships from
identifiers	A list of the identifiers of data objects contained in the associated data package
resolveURI	A character string containing a URI to prepend to datapackage identifiers.
externalIdentifiers	A list of identifiers that are referenced from the package, but are not package members.
creator	A character string containing the creator of the package.

Details

The `identifiers` parameter contains the identifiers of all data objects in the DataPackage. For each data objects, additional relationships will be added that are required by the OAI-ORE specification, for example a Dublin Core identifier statement is added. The resolveURI string value is prepended to DataPackage member identifiers in the resulting resource map. If no resolveURI value is specified, then 'https://cn.dataone.org/cn/v1/resolve' is used.

See Also

`ResourceMap-class`

Examples

```
library(datapack)
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
do1 <- new("DataObject", id="id1", data, format="text/csv")
do2 <- new("DataObject", id="id2", data, format="text/csv")
dp <- addMember(dp, do1)
dp <- addMember(dp, do2)
dp <- insertRelationship(dp, subjectID="id1", objectIDs="id2",
  predicate="http://www.w3.org/ns/prov#wasDerivedFrom")
relations <- getRelationships(dp)
resMapId <- sprintf("%s%s", "resourceMap_", uuid::UUIDgenerate())
resMap <- new("ResourceMap", id=resMapId)
resMap <- createFromTriples(resMap, relations, getIdentifiers(dp))
```

DataObject-class *DataObject wraps raw data with system-level metadata*

Description

DataObject is a wrapper class that associates raw data or a data file with system-level metadata describing the data. The system metadata includes attributes such as the object's identifier, type, size, checksum, owner, version relationship to other objects, access rules, and other critical metadata. The SystemMetadata is compliant with the DataONE federated repository network's definition of SystemMetadata, and is encapsulated as a separate object of type `SystemMetadata` that can be manipulated as needed. Additional science-level and domain-specific metadata is out-of-scope for SystemMetadata, which is intended only for critical metadata for managing objects in a repository system.

Details

A DataObject can be constructed by passing the data and SystemMetadata to the new() method, or by passing an identifier, data, format, user, and DataONE node identifier, in which case a SystemMetadata instance will be generated with these fields and others that are calculated (such as size and checksum).

Data are associated with the DataObject either by passing it as a 'raw' value to the 'dataobj' parameter in the constructor, which is then stored in memory, or by passing a fully qualified file path to the data in the 'filename' parameter, which is then stored on disk. One of dataobj or filename is required. Use the 'filename' approach when data are too large to be managed effectively in memory. Callers can access the 'filename' slot to get direct access to the file, or can call 'getData()' to retrieve the contents of the data or file as a raw value (but this will read all of the data into memory).

Slots

`sysmeta` A value of type "SystemMetadata", containing the metadata about the object

`data` A value of type "raw", containing the data represented in this object

`filename` A character value that contains the fully-qualified path to the object data on disk

`dataURL` A character value for the URL used to load data into this DataObject

`updated` A list containing logical values which indicate if system metadata or the data object have been updated since object creation.

`oldId` A character string containing the previous identifier used, before a "replaceMember" call.

`targetPath` An optional character string holding the path of where the file is placed in a downloaded package.

Methods

`initialize` : Initialize a DataObject

`addAccessRule` : Add a Rule to the AccessPolicy

`canRead` : Test whether the provided subject can read an object.

`getData` : Get the data content of a specified data object

`getFormatId` : Get the FormatId of the DataObject

`getIdentifier` : Get the Identifier of the DataObject

`hasAccessRule` : Determine if an access rules exists for a DataObject.

`setPublicAccess` : Add a Rule to the AccessPolicy to make the object publicly readable.

`updateXML` : Update selected elements of the xml content of a DataObject

See Also

`datapack`

Examples

```
data <- charToRaw("1,2,3\n4,5,6\n")
targetPath <- "myData/time-trials/trial_data.csv"
do <- new("DataObject", "id1", dataobj=data, "text/csv",
  "uid=jones,DC=example,DC=com", "urn:node:KNB", targetPath=targetPath)
getIdentifier(do)
getFormatId(do)
getData(do)
canRead(do, "uid=anybody,DC=example,DC=com")
```

```
do <- setPublicAccess(do)
canRead(do, "public")
canRead(do, "uid=anybody,DC=example,DC=com")
# Also can create using a file for storage, rather than memory
## Not run:
tf <- tempfile()
con <- file(tf, "wb")
writeBin(data, con)
close(con)
targetPath <- "myData/time-trials/trial_data.csv"
do <- new("DataObject", "id1", format="text/csv", user="uid=jones,DC=example,DC=com",
  mnNodeId="urn:node:KNB", filename=tf, targetPath=targetPath)

## End(Not run)
```

`DataPackage-class` *A class representing a data package*

Description

The DataPackage class provides methods for adding and extracting data objects from a data package. The contents of a data package can include arbitrary types of objects, including data files, program code, visualizations and images, animations, and any other type of file. The DataPackage class stores the individual members of the data package along with key system-level metadata about each object, including its size, checksum, identifier, and other key information needed to effectively archive the members of the package. In addition, the DataPackage class can include key provenance metadata about the relationships among the objects in the data package. For example, the data package can document that one object provides documentation for another (`cito:documents`), and that one object was derived from another (`prov:wasDerivedFrom`) by executing a program that used source data (`prov:used`) to create a derived data object {`prov:wasGeneratedBy`}. These relationships are integral to the data package, and can be visualized by programs that understand the ProvONE provenance model (see https://purl.dataone.org/provone-v1-dev).

The DataPackage class is an R representation of an underlying Open Archives Initiative ORE model (Object Reuse and Exchange; see https://www.openarchives.org/ore/), and follows the DataONE Data Packaging model (see https://releases.dataone.org/online/api-documentation-v2.0.1/design/DataPackage.html).

Slots

`relations` A list containing provenance relationships of package objects

`objects` A list containing identifiers for objects in the DataPackage

`sysmeta` A SystemMetadata class instance describing the package

`externalIds` A list containing identifiers for objects associated with the DataPackage

`resmapId` A character string specifying the identifier for the package resource map. This is assigned after a package is uploaded or downloaded from a repository.

Methods

initialize : Initialize a DataPackage object.

addAccessRule : Add access rules to DataObjects in a DataPackage.

addMember : Add a DataObject to a DataPackage.

clearAccessPolicy : Clear access policies for DataObjects in a DataPackage.

containsId : Returns true if the specified object is a member of the data package.

describeWorkflow : Add data derivation information to a DataPackage.

getData : Get the data content of a specified data object.

getSize : Get the Count of Objects in the DataPackage.

getIdentifiers : Get the Identifiers of DataPackage members.

getMember : Return the DataPackage Member by Identifier.

getRelationships : Retrieve relationships of data package objects.

getValue : Get values for selected DataPackage members.

hasAccessRule : Determine if access rules exists for DataObjects in a DataPackage.

insertRelationship : Insert relationships between objects in a DataPackage.

removeAccessRule : Remove an access rule from DataObject in a DataPackage.

removeMember : Remove the specified DataObject from a DataPackage.

removeRelationships : Remove relationships of objects in a DataPackage.

replaceMember : Replace the raw data or file associated with a DataObject.

selectMember : Select package members based on slot values.

serializePackage : Create an OAI-ORE resource map from the DataPackage.

serializeToBagIt : Serialize A DataPackage into a BagIt Archive File.

setPublicAccess : Set the access policy to readable by anyone for DataObject in a DataPackage.

setValue : Set values for selected DataPackage members

show : Print DataPackage information in a formatted view.

updateMetadata : Update selected elements of the XML content of a DataObject in a DataPackage

updateRelationships : Update package relationships by replacing an old identifier with a new one.

See Also

datapack

describeWorkflow *Add data derivation information to a DataPackage*

Description

Add information about the relationships among DataObject members in a DataPackage, retrospectively describing the way in which derived data were created from source data using a processing program such as an R script. These provenance relationships allow the derived data to be understood sufficiently for users to be able to reproduce the computations that created the derived data, and to trace lineage of the derived data objects. The method `describeWorkflow` will add provenance relationships between a script that was executed, the files that it used as sources, and the derived files that it generated.

Usage

```
describeWorkflow(x, ...)

## S4 method for signature 'DataPackage'
describeWorkflow(
  x,
  sources = list(),
  program = NA_character_,
  derivations = list(),
  insertDerivations = TRUE,
  ...
)
```

Arguments

`x`	The `DataPackage` to add provenance relationships to.
`...`	Additional parameters
`sources`	A list of DataObjects for files that were read by the program. Alternatively, a list of DataObject identifiers can be specified as a list of character strings.
`program`	The DataObject created for the program such as an R script. Alternatively the DataObject identifier can be specified.
`derivations`	A list of DataObjects for files that were generated by the program. Alternatively, a list of DataObject identifiers can be specified as a list of character strings.
`insertDerivations`	A `logical` value. If TRUE then the provenance relationship `prov:wasDerivedFrom` will be used to connect every source and derivation. The default value is TRUE.

Details

This method operates on a DataPackage that has had DataObjects for the script, data sources (inputs), and data derivations (outputs) previously added to it, or can reference identifiers for objects that exist in other DataPackage instances. This allows a user to create a standalone package that

contains all of its source, script, and derived data, or a set of data packages that are chained together via a set of derivation relationships between the members of those packages.

Provenance relationships are described following the the ProvONE data model, which can be viewed at https://purl.dataone.org/provone-v1-dev. In particular, the following relationships are inserted (among others):

`prov:used` indicates which source data was used by a program execution

`prov:generatedBy` indicates which derived data was created by a program execution

`prov:wasDerivedFrom` indicates the source data from which derived data were created using the program

See Also

The R 'recordr' package for run-time recording of provenance relationships.

Examples

```
library(datapack)
dp <- new("DataPackage")
# Add the script to the DataPackage
progFile <- system.file("./extdata/pkg-example/logit-regression-example.R", package="datapack")
progObj <- new("DataObject", format="application/R", filename=progFile)
dp <- addMember(dp, progObj)

# Add a script input to the DataPackage
inFile <- system.file("./extdata/pkg-example/binary.csv", package="datapack")
inObj <- new("DataObject", format="text/csv", filename=inFile)
dp <- addMember(dp, inObj)

# Add a script output to the DataPackage
outFile <- system.file("./extdata/pkg-example/gre-predicted.png", package="datapack")
outObj <- new("DataObject", format="image/png", file=outFile)
dp <- addMember(dp, outObj)

# Add the provenenace relationshps, linking the input and output to the script execution
# Note: 'sources' and 'derivations' can also be lists of "DataObjects" or "DataObject' identifiers
dp <- describeWorkflow(dp, sources = inObj, program = progObj, derivations = outObj)
# View the results
utils::head(getRelationships(dp))
```

`dmsg` *Print a debugging message to stderr*

Description

Print a debugging message to stderr

Usage

```
dmsg(msg)
```

Arguments

msg	the message to be printed

Details

Only print the message if the option "datapack.debugging_mode" is TRUE.

---

freeResourceMap *Free memory used by a ResouceMap*

Description

The resources allocated by the redland RDF package are freed. The ResourceMap object should be deleted immediately following this call.

Usage

```
freeResourceMap(x)

## S4 method for signature 'ResourceMap'
freeResourceMap(x)
```

Arguments

x	a ResourceMap

See Also

ResourceMap-class

---

getData *Get the data content of a specified data object*

Description

Get the data content of a specified data object

Usage

```
getData(x, ...)

## S4 method for signature 'DataObject'
getData(x)

## S4 method for signature 'DataPackage'
getData(x, id)
```

Arguments

x	DataObject or DataPackage: the data structure from where to get the data
...	Additional arguments
id	Missing or character: if 'x' is DataPackage, the identifier of the package member to get data from

Value

raw representation of the data

See Also

DataObject-class

Examples

```
data <- charToRaw("1,2,3\n4,5,6\n")
do <- new("DataObject", "id1", dataobj=data, "text/csv",
  "uid=jones,DC=example,DC=com", "urn:node:KNB")
bytes <- getData(do)
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
do1 <- new("DataObject", id="id1", data, format="text/csv", user="smith", mnNodeId="urn:node:KNB")
dp <- addMember(dp, do1)
bytes <- getData(dp, "id1")
```

---

getFormatId *Get the FormatId of the DataObject*

Description

Get the FormatId of the DataObject

Usage

```
getFormatId(x, ...)

## S4 method for signature 'DataObject'
getFormatId(x)
```

Arguments

x	DataObject
...	(not yet used)

Value

the formatId

See Also

DataObject-class

Examples

```
data <- charToRaw("1,2,3\n4,5,6\n")
do <- new("DataObject", "id1", dataobj=data, "text/csv",
  "uid=jones,DC=example,DC=com", "urn:node:KNB")
fmtId <- getFormatId(do)
```

---

getIdentifier *Get the Identifier of the DataObject*

Description

Get the Identifier of the DataObject

Usage

```
getIdentifier(x, ...)

## S4 method for signature 'DataObject'
getIdentifier(x)
```

Arguments

x	DataObject
...	(not yet used)

Value

the identifier

See Also

DataObject-class

Examples

```
data <- charToRaw("1,2,3\n4,5,6\n")
do <- new("DataObject", "id1", dataobj=data, "text/csv",
  "uid=jones,DC=example,DC=com", "urn:node:KNB")
id <- getIdentifier(do)
```

getIdentifiers *Get the Identifiers of Package Members*

Description

The identifiers of the objects in the package are retrieved and returned as a list.

Usage

```
getIdentifiers(x, ...)

## S4 method for signature 'DataPackage'
getIdentifiers(x)
```

Arguments

x	A DataPackage instance
...	(not yet used)

Value

A list of identifiers

See Also

DataPackage-class

Examples

```
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
do <- new("DataObject", dataobj=data, format="text/csv", user="jsmith")
dp <- addMember(dp, do)
getIdentifiers(dp)
```

getMember *Return the Package Member by Identifier*

Description

Given the identifier of a member of the data package, return the DataObject representation of the member.

Usage

```
getMember(x, ...)

## S4 method for signature 'DataPackage'
getMember(x, identifier)
```

Arguments

x	A DataPackage instance
...	(Not yet used)
identifier	A DataObject identifier

Value

A DataObject if the member is found, or NULL if not

See Also

DataPackage-class

Examples

```
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
do <- new("DataObject", id="myNewId", dataobj=data, format="text/csv", user="jsmith")
dp <- addMember(dp, do)
do2 <- getMember(dp, "myNewId")
```

---

getRelationships *Retrieve relationships of package objects*

Description

Relationships of objects in a package are defined using the 'insertRelationship' call and retrieved using getRetaionships. These relationships are returned in a data frame with 'subject', 'predicate', 'objects' as the columns, ordered by "subject"

Usage

```
getRelationships(x, ...)

## S4 method for signature 'DataPackage'
getRelationships(x, condense = F, ...)
```

Arguments

x	A DataPackage object
...	(Not yet used)
condense	A logical value, if TRUE then a more easily viewed version of relationships are returned.

See Also

DataPackage-class

Examples

```
dp <- new("DataPackage")
insertRelationship(dp, "/Users/smith/scripts/genFields.R",
    "http://www.w3.org/ns/prov#used",
   "https://knb.ecoinformatics.org/knb/d1/mn/v1/object/doi:1234/_030MXTI009R00_20030812.40.1")
rels <- getRelationships(dp)
```

---

getSize *Get the Count of Objects in the Package*

Description

Get the Count of Objects in the Package

Usage

```
getSize(x, ...)

## S4 method for signature 'DataPackage'
getSize(x)
```

Arguments

x	A DataPackage instance
...	(not yet used)

Value

The number of object in the Package

See Also

DataPackage-class

Examples

```
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
do <- new("DataObject", dataobj=data, format="text/csv", user="jsmith")
dp <- addMember(dp, do)
getSize(dp)
```

`getTriples` *Get the RDF relationships stored in the ResourceMap*

Description

The `getTriples` method extracts the RDF relationships from a ResourceMap.

Usage

```
getTriples(x, ...)

## S4 method for signature 'ResourceMap'
getTriples(x, filter = TRUE, identifiers = list(), ...)
```

Arguments

`x`	ResourceMap
`...`	Additional parameters (not yet implemented).
`filter`	A `logical` value. If TRUE, then DataONE packaging relationships are omitted.
`identifiers`	A list of `character` values of the identifiers of DataPackage members.

Details

The `filter` argument causes DataONE packaging relationships to be removed. A description of these can be viewed at https://purl.dataone.org/architecture/design/DataPackage.html. The `identifiers` parameter can contain a list of DataPackage members for which the identifiers will be 'demoted', that is any relationship that has these identifiers as a URL as the subject or object will be changed to the 'bare' identifier. The intent of these two parameter is to transform the DataPackage to a 'local' state, so that it can be more easily updated locally.

Value

x A data.frame containing the relationships from the ResourceMap

getValue *Get values for selected DataPackage members*

Description

Given a slot name and set of package member identifiers, return slot values.

Usage

```
getValue(x, ...)

## S4 method for signature 'DataPackage'
getValue(x, name, identifiers = NA_character_)
```

Arguments

x	A DataPackage instance
...	(Not yet used)
name	A name of a DataObject slot.
identifiers	A list of DataPackage member identifiers

Details

If the parameter `identifiers` is provided, then only the DataPackage members that have identifiers in the provided list will have there values fetched. If this parameter is not provided, then the values for all DataPackage members are returned.

Value

A list of values for matching slot names and included identifiers.

See Also

DataPackage-class

Examples

```
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
do <- new("DataObject", id="myNewId", dataobj=data, format="text/csv", user="jsmith")
dp <- addMember(dp, do)
data <- charToRaw("7,8.9\n4,10,11")
do <- new("DataObject", id="myNewId2", dataobj=data, format="text/csv", user="jsmith")
dp <- addMember(dp, do)
formats <- getValue(dp, name="sysmeta@formatId")
```

`hasAccessRule` *Determine if an access rules exists*

Description

Each SystemMetadata document may contain a set of (subject, permission) tuples that represent the access rules for its associated object. This method determines whether a particular access rule already exists within the set.

If called for a DataObject, then the SystemMetadata for the DataObject is checked.

If called for a DataPackage, then the SystemMetadata for DataObjects in the DataPackage are checked.

Usage

```
hasAccessRule(x, ...)

## S4 method for signature 'SystemMetadata'
hasAccessRule(x, subject, permission)

## S4 method for signature 'DataObject'
hasAccessRule(x, subject, permission)

## S4 method for signature 'DataPackage'
hasAccessRule(x, subject, permission, identifiers = list(), ...)
```

Arguments

`x`	the object to check for presence of the access rule.
`...`	Additional arguments
`subject`	of the rule to be checked
`permission`	the permission to be checked
`identifiers`	A list of `character` values containing package member identifiers for which the access rule will be checked.

Value

A logical value: if TRUE the access rule was found, if FALSE it was not found.

When called for SystemMetadata, boolean TRUE if the access rule exists already, FALSE otherwise

When called for a DataObject, boolean TRUE if the access rule exists already, FALSE otherwise

When called for a DataPackage, boolean TRUE if the access rule exists in all specified package members already, FALSE otherwise

See Also

SystemMetadata-class

DataObject-class

DataPackage-class

Examples

```
#
# Check access rules for a SystemMetadata object.
sysmeta <- new("SystemMetadata")
sysmeta <- addAccessRule(sysmeta, "uid=smith,ou=Account,dc=example,dc=com", "write")
accessRules <- data.frame(subject=c("uid=smith,ou=Account,dc=example,dc=com",
 "uid=slaughter,o=unaffiliated,dc=example,dc=org"), permission=c("write", "changePermission"))
sysmeta <- addAccessRule(sysmeta, accessRules)
ruleExists <- hasAccessRule(sysmeta, subject="uid=smith,ou=Account,dc=example,dc=com",
  permission="write")
#
# Check access rules for a DataObject
data <- system.file("extdata/sample-data.csv", package="datapack")
do <- new("DataObject", file=system.file("./extdata/sample-data.csv", package="datapack"),
                                        format="text/csv")
do <- setPublicAccess(do)
isPublic <- hasAccessRule(do, "public", "read")
accessRules <- data.frame(subject=c("uid=smith,ou=Account,dc=example,dc=com",
                          "uid=wiggens,o=unaffiliated,dc=example,dc=org"),
                          permission=c("write", "changePermission"),
                          stringsAsFactors=FALSE)
do <- addAccessRule(do, accessRules)
SmithHasWrite <- hasAccessRule(do, "uid=smith,ou=Account,dc=example,dc=com", "write")
#
# Check access rules for member DataObjects of a DataPackage.
# First create an example DataPackage
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6\n")
obj <- new("DataObject", id="id1", dataobj=data, format="text/csv")
dp <- addMember(dp, obj)
data2 <- charToRaw("7,8,9\n4,10,11\n")
obj2 <- new("DataObject", id="id2", dataobj=data2, format="text/csv")
dp <- addMember(dp, obj2)
# Add access rules to all package members
dp <- addAccessRule(dp, "uid=smith,ou=Account,dc=example,dc=com", "write")
dp <- addAccessRule(dp, "uid=smith,ou=Account,dc=example,dc=com", "changePermission")
hasWrite <- hasAccessRule(dp, "uid=smith,ou=Account,dc=example,dc=com", "write")
hasChange <- hasAccessRule(dp, "uid=smith,ou=Account,dc=example,dc=com", "changePermission")
```

---

initialize,DataObject-method

Initialize a DataObject

Description

When initializing a DataObject using passed in data, one can either pass in the 'id' param as a 'SystemMetadata' object, or as a 'character' string representing the identifier for an object along with parameters for format, user,and associated member node. If 'data' is not missing, the 'data' param holds the 'raw' data. Otherwise, the 'filename' parameter must be provided, and points at a file containing the bytes of the data.

Usage

```
## S4 method for signature 'DataObject'
initialize(
  .Object,
  id = NA_character_,
  dataobj = NA,
  format = NA_character_,
  user = NA_character_,
  mnNodeId = NA_character_,
  filename = NA_character_,
  seriesId = NA_character_,
  mediaType = NA_character_,
  mediaTypeProperty = list(),
  dataURL = NA_character_,
  targetPath = NA_character_,
  checksumAlgorithm = "SHA-256"
)
```

Arguments

.Object	the DataObject instance to be initialized
id	the identifier for the DataObject, unique within its repository. Optionally this can be an existing SystemMetadata object
dataobj	the bytes of the data for this object in 'raw' format, optional if 'filename' is provided
format	the format identifier for the object, e.g."text/csv", "eml://ecoinformatics.org/eml-2.1.1"
user	the identity of the user owning the package, typically in X.509 format
mnNodeId	the node identifier for the repository to which this object belongs.
filename	the filename for the fully qualified path to the data on disk, optional if 'data' is provided
seriesId	A unique string to identifier the latest of multiple revisions of the object.
mediaType	The When specified, indicates the IANA Media Type (aka MIME-Type) of the object. The value should include the media type and subtype (e.g. text/csv).
mediaTypeProperty	A list, indicates IANA Media Type properties to be associated with the parameter "mediaType"

dataURL	A character string containing a URL to remote data (a repository) that this DataObject represents.
targetPath	An optional string that denotes where the file should go in a downloaded package
checksumAlgorithm	A character string specifying the checksum algorithm to use

Details

If filesystem storage is used for the data associated with a DataObject, care must be taken to not modify or remove that file in R or via other facilities while the DataObject exists in the R session. Changes to the object are not detected and will result in unexpected results. Also, if the 'dataobj' parameter is used to specify the data source, then 'filename' argument may also be specified, but in this case the value 'filename' parameter is used to tell DataONE the filename to create when this file is downloaded from a repository.

See Also

DataObject-class

Examples

```
data <- charToRaw("1,2,3\n4,5,6\n")
do <- new("DataObject", "id1", dataobj=data, "text/csv",
  "uid=jones,DC=example,DC=com", "urn:node:KNB", targetPath="data/rasters/data.tiff")
```

---

initialize,DataPackage-method

Initialize a DataPackage object

Description

Initialize a DataPackage object

Usage

```
## S4 method for signature 'DataPackage'
initialize(.Object, packageId)
```

Arguments

.Object	The object being initialized
packageId	The package id to assign to the package

See Also

DataPackage-class

Examples

```
# Create a DataPackage with undefined package id (to be set manually later)
pkg <- new("DataPackage")
# Alternatively, manually assign the package id when the DataPackage object is created
pkg <- new("DataPackage", "urn:uuid:4f953288-f593-49a1-adc2-5881f815e946")
```

`initialize,ResourceMap-method`

Initialize a ResourceMap object

Description

Create a ResourceMap object that contains relationships (RDF triples) of objects in the DataPackage.

Usage

```
## S4 method for signature 'ResourceMap'
initialize(.Object, id = NA_character_)
```

Arguments

`.Object`	a ResourceMap object
`id`	a unique identifier to identify this ResourceMap. This id will be used internally in the ResourceMap.

Value

the ResourceMap object

See Also

`ResourceMap-class`

`initialize,SystemMetadata-method`

Initialize a DataONE SystemMetadata object with default values or values passed in to the constructor

---

Description

Initialize a SystemMetadata object by providing default values for core information needed to manage objects across repository systems. SystemMetadata contains basic identification, ownership, access policy, replication policy, and related metadata.

Usage

```
## S4 method for signature 'SystemMetadata'
initialize(
  .Object,
  identifier = NA_character_,
  formatId = NA_character_,
  size = NA_real_,
  checksum = NA_character_,
  checksumAlgorithm = "SHA-256",
  submitter = NA_character_,
  rightsHolder = NA_character_,
  accessPolicy = data.frame(subject = character(), permission = character()),
  replicationAllowed = TRUE,
  numberReplicas = 3,
  obsoletes = NA_character_,
  obsoletedBy = NA_character_,
  archived = FALSE,
  dateUploaded = NA_character_,
  dateSysMetadataModified = NA_character_,
  originMemberNode = NA_character_,
  authoritativeMemberNode = NA_character_,
  preferredNodes = list(),
  blockedNodes = list(),
  seriesId = NA_character_,
  mediaType = NA_character_,
  fileName = NA_character_,
  mediaTypeProperty = list()
)
```

Arguments

`.Object`	The object being initialized
`identifier`	value of type `"character"`, the identifier of the object that this system metadata describes.
`formatId`	value of type `"character"`, the DataONE object format for the object.
`size`	value of type `"numeric"`, the size of the object in bytes.
`checksum`	value of type `"character"`, the checksum for the object using the designated checksum algorithm.
`checksumAlgorithm`	value of type `"character"`, the name of the hash function used to generate a checksum, from the DataONE controlled list.
`submitter`	value of type `"character"`, the Distinguished Name or identifier of the person submitting the object.
`rightsHolder`	value of type `"character"`, the Distinguished Name or identifier of the person who holds access rights to the object.

`accessPolicy` value of type `"data.frame"` containing (subject, permission) tuples to constitute the access authorization rules.

`replicationAllowed` value of type `"logical"`, for replication policy allows replicas.

`numberReplicas` value of type `"numeric"`, for number of supported replicas.

`obsoletes` value of type `"character"`, the identifier of an object which this object replaces.

`obsoletedBy` value of type `"character"`, the identifier of an object that replaces this object.

`archived` value of type `"logical"`, a boolean flag indicating whether the object has been archived and thus hidden.

`dateUploaded` value of type `"character"`, the date on which the object was uploaded to a member node.

`dateSysMetadataModified` value of type `"character"`, the last date on which this system metadata was modified.

`originMemberNode` value of type `"character"`, the node identifier of the node on which the object was originally registered.

`authoritativeMemberNode` value of type `"character"`, the node identifier of the node which currently is authoritative for the object.

`preferredNodes` list of `"character"`, each of which is the node identifier for a node to which a replica should be sent.

`blockedNodes` list of `"character"`, each of which is the node identifier for a node blocked from housing replicas.

`seriesId` value of type `"character"`, a unique Unicode string that identifies an object revision chain. A seriesId will resolve to the latest version of an object.

`mediaType` value of type `"character"`, the IANA Media Type (aka MIME-Type) of the object, e.g. "text/csv".

`fileName` value of type `"character"`, the name of the file to create when this object is downloaded from DataONE.

`mediaTypeProperty` value of type a `"list"` of `"character"`, IANA Media Type properties for the `"mediaType"` argument

Value

the SystemMetadata instance representing an object

See Also

https://releases.dataone.org/online/api-documentation-v2.0/apis/Types.html

SystemMetadata-class

`insertRelationship` *Record relationships of objects in a DataPackage*

Description

Record a relationship of the form "subject -> predicate -> object", as defined by the Resource Description Framework (RDF), i.e. an RDF triple.

Usage

```
insertRelationship(x, ...)

## S4 method for signature 'DataPackage'
insertRelationship(
  x,
  subjectID,
  objectIDs,
  predicate = NA_character_,
  subjectType = NA_character_,
  objectTypes = NA_character_,
  dataTypeURIs = NA_character_
)
```

Arguments

`x`	A DataPackage object
`...`	(Additional parameters)
`subjectID`	The identifier of the subject of the relationship
`objectIDs`	A list of identifiers of the object of the relationships (a relationship is recorded for each objectID)
`predicate`	The IRI of the predicate of the relationship
`subjectType`	the type to assign the subject, values can be 'uri', 'blank'
`objectTypes`	the types to assign the objects (cal be single value or list), each value can be 'uri', 'blank', or 'literal'
`dataTypeURIs`	An RDF data type that specifies the type of the object

Details

For use with DataONE, a best practice is to specify the subject and predicate as DataONE persistent identifiers (https://mule1.dataone.org/ArchitectureDocs-current/design/PIDs.html). If the objects are not known to DataONE, then local identifiers can be used, and these local identifiers may be promoted to DataONE PIDs when the package is uploaded to a DataONE member node. The predicate is typically an RDF property (as a IRI) from a schema supported by DataONE, i.e. "http://www.w3.org/ns/prov#wasGeneratedBy" If multiple values are specified for argument objectIDS, a relationship is created for each value in the list "objectIDs". IF a value is not specified for

subjectType or objectType, then NA is assigned. Note that if these relationships are fetched via the getRelationships() function, and passed to the createFromTriples() function to initialize a ResourceMap object, the underlying redland package will assign appropriate values for subjects and objects. Note: This method updates the passed-in DataPackage object.

Value

the updated DataPackage object

See Also

DataPackage-class

Examples

```
dp <- new("DataPackage")
# Create a relationship
dp <- insertRelationship(dp, "/Users/smith/scripts/genFields.R",
   "https://knb.ecoinformatics.org/knb/d1/mn/v1/object/doi:1234/_030MXTI009R00_20030812.40.1",
    "http://www.w3.org/ns/prov#used")
# Create a relationshp with the subject as a blank node with an automatically assigned blank
# node id
dp <- insertRelationship(dp, subjectID=NA_character_, objectIDs="thing6",
    predicate="http://www.myns.org/wasThing")
# Create a relationshp with the subject as a blank node with a user assigned blank node id
dp <- insertRelationship(dp, subjectID="urn:uuid:bc9e160e-ca21-47d5-871b-4a4820fe4451",
      objectIDs="thing7", predicate="http://www.myns.org/hadThing")
# Create multiple relationships with the same subject, predicate, but different objects
dp <- insertRelationship(dp, subjectID="urn:uuid:95055dc1-b2a0-4a00-bdc2-05c16d048ca2",
      objectIDs=c("thing4", "thing5"), predicate="http://www.myns.org/hadThing")
# Create multiple relationships with subject and object types specified
dp <- insertRelationship(dp, subjectID="orcid.org/0000-0002-2192-403X",
    objectIDs="http://www.example.com/home", predicate="http://www.example.com/hadHome",
                  subjectType="uri", objectType="literal")
```

---

parseRDF *Parse an RDF/XML resource map from a file*

Description

parseRDF reads a file containing an RDF model in RDF/XML format and initializes a ResourceMap based on this content.

Usage

```
parseRDF(x, rdf, ...)

## S4 method for signature 'ResourceMap'
parseRDF(
```

```
  x,
  rdf,
  asText = FALSE,
  name = "rdfxml",
  mimeType = "application/rdf+xml",
  ...
)
```

Arguments

x	ResourceMap
rdf	A file or character value containing a resource map that will be parsed into the ResourceMap object
...	Additional parameters (not yet used).
asText	A logical value. If TRUE, then the 'rdf' parameter is a character vector, if FALSE then it is the name of a file to read.
name	The name of the RDF xml parser, the default is "rdfxml".
mimeType	A character value containing the RDF format type. The default is "application/rdf+xml".

Details

This method resets the slot ResourceMap@world so any previously stored triples are discarded, allowing for a clean model object in which to parse the new RDF content into. It is assumed that the content is a valid ORE resource map therefor no validation checks specific to the OAI-ORE content model are performed.

Value

x the ResourceMap containing the parsed RDF/XML content

---

parseSystemMetadata *Parse an external XML document and populate a SystemMetadata object with the parsed data*

Description

Parse an XML representation of system metadata, and set the object slots of a SystemMetadata object the with obtained values.

Usage

```
parseSystemMetadata(x, ...)

## S4 method for signature 'SystemMetadata'
parseSystemMetadata(x, xml, ...)
```

Arguments

x	The SystemMetadata object
...	Additional arguments passed to other functions or methods
xml	The XML representation of the capabilities, as an XMLInternalElementNode

Value

the SystemMetadata object representing an object

See Also

SystemMetadata-class

Examples

```
library(XML)
doc <- xmlParseDoc(system.file("testfiles/sysmeta.xml", package="datapack"), asText=FALSE)
sysmeta <- new("SystemMetadata")
sysmeta <- parseSystemMetadata(sysmeta, xmlRoot(doc))
```

---

plotRelationships *Plot derivation relationships obtained from getRelationships*

Description

Creates graph of dataPackage object generated from getRelationships

Usage

```
plotRelationships(x, ...)

## S4 method for signature 'DataPackage'
plotRelationships(x, col = NULL, ...)
```

Arguments

x	a DataPackage object
...	other options passed to the igraph plot function
col	vector of colors used for plotting

See Also

DataPackage-class

recordDerivation *Record derivation relationships between objects in a DataPackage*

Description

Record a derivation relationship that expresses that a target object has been derived from a source object. For use with DataONE, a best practice is to specify the subject and predicate as DataONE persistent identifiers (https://mule1.dataone.org/ArchitectureDocs-current/design/PIDs.html). If the objects are not known to DataONE, then local identifiers can be used, and these local identifiers may be promoted to DataONE PIDs when the package is uploaded to a DataONE member node.

Usage

```
recordDerivation(x, ...)

## S4 method for signature 'DataPackage'
recordDerivation(x, sourceID, derivedIDs, ...)
```

Arguments

x	a DataPackage object
...	Additional parameters
sourceID	the identifier of the source object in the relationship
derivedIDs	an identifier or list of identifiers of objects that were derived from the source

Details

A derived relationship is created for each value in the list "objectIDs". For each derivedId, one statement will be added expressing that it was derived from the sourceId. The predicate is will be an RDF property (as a IRI) from the W3C PROV specification, namely, "http://www.w3.org/ns/prov#wasDerivedFrom"

See Also

DataPackage-class

Examples

```
## Not run:
dp <- new("DataPackage")
recordDerivation(dp, "doi:1234/_030MXTI009R00_20030812.40.1",
                 "doi:1234/_030MXTI009R00_20030812.45.1")

## End(Not run)
```

`removeAccessRule` *Remove an access rule from the specified object*

Description

Remove access rules from the access policy of the specified object.

Usage

```
removeAccessRule(x, ...)

## S4 method for signature 'SystemMetadata'
removeAccessRule(x, y, ...)

## S4 method for signature 'DataObject'
removeAccessRule(x, y, ...)

## S4 method for signature 'DataPackage'
removeAccessRule(x, y, permission = NA_character_, identifiers = list(), ...)
```

Arguments

`x`	The object instance to which to remove the rule
`...`	Additional arguments • `permission`: The permission to be remove to subject if x is character (read, write, changePermission)
`y`	The subject of the rule to be removed, or a data.frame containing access rules.
`permission`	The permission to remove, if parameter `x` is a character string containing a `subject`.
`identifiers`	A list of `character` values containing package member identifiers that the access rule will be applied to (default is all package members).

Value

The SystemMetadata object with the updated access policy.

The DataObject object with the updated access policy.

The Datapackage with members having updated access policies.

See Also

`SystemMetadata-class`

`DataObject-class`

`DataPackage-class`

Examples

```
#
# Remove access rules from a SystemMetadata object.
# Parameter "y" can be character string containing the subject of the access rule:
sysmeta <- new("SystemMetadata")
sysmeta <- addAccessRule(sysmeta, "uid=smith,ou=Account,dc=example,dc=com", "write")
sysmeta <- addAccessRule(sysmeta, "uid=smith,ou=Account,dc=example,dc=com", "changePermission")
sysmeta <- removeAccessRule(sysmeta, "uid=smith,ou=Account,dc=example,dc=com", "changePermission")

# Alternatively, parameter "y" can be a data.frame containing one or more access rules:
# Add write, changePermission for uid=jones,...
sysmeta <- addAccessRule(sysmeta, "uid=jones,ou=Account,dc=example,dc=com", "write")
sysmeta <- addAccessRule(sysmeta, "uid=jones,ou=Account,dc=example,dc=com", "changePermission")
# Now take privs for uid=jones,... away
accessRules <- data.frame(subject=c("uid=jones,ou=Account,dc=example,dc=com",
                                    "uid=jones,ou=Account,dc=example,dc=com"),
                                    permission=c("write", "changePermission"))
sysmeta <- removeAccessRule(sysmeta, accessRules)
#
# Remove access rules form a DataObject.
library(datapack)
do <- new("DataObject", file=system.file("./extdata/sample-data.csv", package="datapack"),
                        format="text/csv")
do <- setPublicAccess(do)
isPublic <- hasAccessRule(do, "public", "read")
accessRules <- data.frame(subject=c("uid=smith,ou=Account,dc=example,dc=com",
                          "uid=wiggens,o=unaffiliated,dc=example,dc=org"),
                          permission=c("write", "changePermission"),
                          stringsAsFactors=FALSE)
do <- addAccessRule(do, accessRules)
do <- removeAccessRule(do, "uid=smith,ou=Account,dc=example,dc=com", "changePermission")
# hasAccessRule should return FALSE
hasWrite <- hasAccessRule(do, "smith", "write")

# Alternatively, parameter "y" can be a data.frame containing one or more access rules:
do <- addAccessRule(do, "uid=smith,ou=Account,dc=example,dc=com", "write")
accessRules <- data.frame(subject=c("uid=smith,ou=Account,dc=example,dc=com",
  "uid=slaughter,o=unaffiliated,dc=example,dc=org"),
  permission=c("write", "changePermission"))
sysmeta <- removeAccessRule(do, accessRules)
#
# Remove access rules from a DataPackage.
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6\n")
obj <- new("DataObject", id="id1", dataobj=data, format="text/csv")
dp <- addMember(dp, obj)
data2 <- charToRaw("7,8,9\n4,10,11\n")
obj2 <- new("DataObject", id="id2", dataobj=data2, format="text/csv")
dp <- addMember(dp, obj2)
# Add access rule to all package members
dp <- addAccessRule(dp, "uid=smith,ou=Account,dc=example,dc=com", "write")
dp <- addAccessRule(dp, "uid=smith,ou=Account,dc=example,dc=com", "changePermission" )
```

```
# Now take 'changePermission' away for user 'uid=smith...', specifying parameter 'y'
# as a character string containing a 'subject'.
dp <- removeAccessRule(dp, "uid=smith,ou=Account,dc=example,dc=com", "write")
dp <- removeAccessRule(dp, "uid=smith,ou=Account,dc=example,dc=com", "changePermission")

# Alternatively, parameter "y" can be a data.frame containing one or more access rules:
# Add write, changePermission for uid=jones,...
dp <- addAccessRule(dp, "uid=jones,ou=Account,dc=example,dc=com", "write")
dp <- addAccessRule(dp, "uid=jones,ou=Account,dc=example,dc=com", "changePermission")
# Now take privs for uid=jones,... away
accessRules <- data.frame(subject=c("uid=jones,ou=Account,dc=example,dc=com",
                                    "uid=jones,ou=Account,dc=example,dc=com"),
                                    permission=c("write", "changePermission"))
dp <- removeAccessRule(dp, accessRules)
```

`removeMember` *Remove the Specified Member from the Package*

Description

Given the identifier of a DataObject in a DataPackage, delete the DataObject from the DataPackage.

Usage

```
removeMember(x, ...)

## S4 method for signature 'DataPackage'
removeMember(x, do, removeRelationships = FALSE)
```

Arguments

`x`	a DataPackage object
`...`	(Not yet used)
`do`	The package member to remove, either as a `"DataObject"` or `"character"` (for the object identifier)
`removeRelationships`	A `logical` value. If TRUE, package relationships for this package member are removed. Default is FALSE.

Details

The `removeMember` method removes the specified DataObject from the DataPackage. In addition, any package relationships that included the DataObject are removed.

See Also

`DataPackage-class`

Examples

```
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
do <- new("DataObject", id="myNewId", dataobj=data, format="text/csv", user="jsmith")
dp <- addMember(dp, do)
# Remove the package member and any provenance relationships that reference it.
removeMember(dp, "myNewId", removeRelationships=TRUE)
```

`removeRelationships` *Remove relationships of objects in a DataPackage*

Description

Use this function to remove all or a subset of the relationships that have previously been added in a data package.

Usage

```
removeRelationships(x, ...)

## S4 method for signature 'DataPackage'
removeRelationships(x, subjectID = NA_character_, predicate = NA_character_)
```

Arguments

`x`	A DataPackage object
`...`	(Additional parameters)
`subjectID`	The identifier of the subject of the relationships to be removed
`predicate`	The identifier of the predicate of the relationships to be removed

Details

Remove a relationship of the form "subject -> predicate -> object", as defined by the Resource Description Framework (RDF), i.e. an RDF triple. If neither subjectID nor predicate are provided, then all relationships are removed. If one or both are provided, they are used to select matching triples to be removed. Note: This method updates the passed-in DataPackage object.

Value

the updated DataPackage object

See Also

DataPackage-class

Examples

```
dp <- new("DataPackage")
# Create a relationship
dp <- insertRelationship(dp, "/Users/smith/scripts/genFields.R",
    "https://knb.org/data_20030812.40.1",
    "http://www.w3.org/ns/prov#used")
# Create a relationshp with the subject as a blank node with an automatically assigned blank
# node id
dp <- insertRelationship(dp, subjectID=NA_character_, objectIDs="thing6",
    predicate="http://myns.org/wasThing")
# Create a relationshp with the subject as a blank node with a user assigned blank node id
dp <- insertRelationship(dp, subjectID="urn:uuid:bc9e160e-ca21-47d5-871b-4a4820fe4451",
      objectIDs="thing7", predicate="http://myns.org/hadThing")
# Create multiple relationships with the same subject, predicate, but different objects
dp <- insertRelationship(dp, subjectID="https://myns.org/subject1",
      objectIDs=c("thing4", "thing5"), predicate="http://myns.org/hadThing")
# Create multiple relationships with subject and object types specified
dp <- insertRelationship(dp, subjectID="orcid.org/0000-0002-2192-403X",
    objectIDs="http://www.example.com/home", predicate="http://myns.org/hadHome",
                  subjectType="uri", objectType="literal")
nrow(getRelationships(dp))
dp <- removeRelationships(dp, predicate='http://myns.org/wasThing')
nrow(getRelationships(dp))
dp <- removeRelationships(dp, subjectID='orcid.org/0000-0002-2192-403X')
nrow(getRelationships(dp))
dp <- removeRelationships(dp, subjectID='https://myns.org/subject1',
    predicate='http://myns.org/hadThing')
nrow(getRelationships(dp))
dp <- removeRelationships(dp)
nrow(getRelationships(dp))
```

`replaceMember` *Replace the raw data or file associated with a DataObject*

Description

A DataObject is a container for data that can be either an R raw object or a file on local disk. The `replaceMember` method can be used to update the date that a DataObject contains, for a DataObject that is a member of a DataPackage, substituting a new file or raw object in the specified DataObject.

Usage

```
replaceMember(x, do, ...)

## S4 method for signature 'DataPackage'
replaceMember(
  x,
  do,
  replacement,
```

```
  formatId = NA_character_,
  mediaType = NA_character_,
  mediaTypeProperty = NA_character_,
  newId = NA_character_,
  ...
)
```

Arguments

x	A DataPackage instance
do	A DataObject instance
...	(Not yet used)
replacement	A raw object or character (for filename) that will replace the current value in the DataObject do.
formatId	A value of type "character", the DataONE object format for the object.
mediaType	A value of type "character", the IANA Media Type (aka MIME-Type) of the object, e.g. "text/csv".
mediaTypeProperty	A value of type "list" of "character", IANA Media Type properties for the "mediaType" argument.
newId	A value of type "character" which will replace the identifier for this DataObject.

Details

The data that is replacing the existing DataObject data may be of a different format or type than the existing data. Because the data type and format may change, the system metadata that describes the data can be updated as well. The replaceMember method will update the SystemMetadata size, checksum values automatically, but does not update the formatId, mediaType, mediaTypeProperty unless requested, so these should be specified in the call to replaceMember if necessary. If the newId argument is used, the specified new identifier will be assigned to the object, otherwise one will be generated if necessary. This new identifier will be used if the DataPackage is uploaded to DataONE, and this object is updating an existing object in DataONE.

See Also

DataPackage-class

Examples

```
# Create a DataObject and add it to the DataPackage
dp <- new("DataPackage")
doIn <- new("DataObject", format="text/csv",
          filename=system.file("./extdata/pkg-example/binary.csv", package="datapack"))
dp <- addMember(dp, doIn)

# Use the zipped version of the file instead by updating the DataObject
dp <- replaceMember(dp, doIn,
```

```
            replacement=system.file("./extdata/pkg-example/binary.csv.zip",
            package="datapack"),
                      formatId="application/zip")
```

`ResourceMap-class` *ResourceMap provides methods to create, serialize and deserialize an OAI ORE resource map*

Description

The Open Archives Initiative Object Reuse and Exchange (OAI-ORE) defines standards for the description and exchange of aggregations of web resources, such as a DataPackage. A Resource Map describes the objects in a DataPackage and the relationships between these objects.

Slots

`relations` value of type `"data.frame"`, containing RDF triples representing the relationship between package objects

`world` a Redland RDF World object

`storage` a Redland RDF Storage object

`model` a Redland RDF Model object

`id` a unique identifier for a ResourceMap instance

Methods

`initialize` : Initialize a ResourceMap object.

`createFromTriples` : Populate a ResourceMap with RDF relationships from data.frame.

`getTriples` : Get the RDF relationships stored in the ResourceMap.

`parseRDF` : Parse an RDF/XML resource map from a file.

`serializeRDF` : Write the ResourceMap relationships to a file.

See Also

`datapack`

Examples

```
dp <- new("DataPackage")
dp <- insertRelationship(dp, "/Users/smith/scripts/genFields.R",
    "http://www.w3.org/ns/prov#used",
  "https://knb.ecoinformatics.org/knb/d1/mn/v1/object/doi:1234/_030MXTI009R00_20030812.40.1")
relations <- getRelationships(dp)
resMap <- new("ResourceMap")
resMap <- createFromTriples(resMap, relations, getIdentifiers(dp))
## Not run:
tf <- tempfile(fileext=".rdf")
```

```
serializeRDF(resMap, file=tf)

## End(Not run)
```

`selectMember` *Return identifiers for objects that match search criteria*

Description

Return DataObjects or DataObject identifiers that match search terms.

Usage

```
selectMember(x, ...)

## S4 method for signature 'DataPackage'
selectMember(x, name, value, as = "character")
```

Arguments

`x`	A DataPackage instance
`...`	(Not yet used)
`name`	The name of the DataObject slot to inspect, for example "sysmeta@formatId".
`value`	A character or logical value to match. If specified as a character value, PERL style regular expressions can be used (see ?grepl).
`as`	A character value to specify the return type, either "DataObject" or "character" (the default)

Details

The `"selectMember"` method inspects the DataObject slot `"name"` for a match with `"value"` for each DataObject in a DataPackage. Matching DataObjects are returned as a list containing either package member identifiers (character) or the DataObjects themselves, depending on the value of the `as` parameter.

Value

A list of matching DataObjects or DataObject identifiers. The default is to return a list of DataObject identifiers.

See Also

`DataPackage-class`

Examples

```
#' library(datapack)
dp <- new("DataPackage")
# Add the script to the DataPackage
progFile <- system.file("./extdata/pkg-example/logit-regression-example.R", package="datapack")
# An 'id' parameter is not specified, so one will be generated automatically.
progObj <- new("DataObject", format="application/R", filename=progFile)
dp <- addMember(dp, progObj)

# Add a script input to the DataPackage
inFile <- system.file("./extdata/pkg-example/binary.csv", package="datapack")
inObj <- new("DataObject", format="text/csv", filename=inFile)
dp <- addMember(dp, inObj)

# Add a script output to the DataPackage
outFile <- system.file("./extdata/pkg-example/gre-predicted.png", package="datapack")
outObj <- new("DataObject", format="image/png", file=outFile)
dp <- addMember(dp, outObj)

# Now determine the package member identifier for the R script
progIds  <- selectMember(dp, name="sysmeta@formatId", value="application/R", as="character")
inputId <- selectMember(dp, name="sysmeta@fileName", value="binary.csv")
```

---

serializePackage *Create an OAI-ORE resource map from the package*

Description

The DataPackage is serialized as a OAI-ORE resource map to the specified file.

Usage

```
serializePackage(x, ...)

## S4 method for signature 'DataPackage'
serializePackage(
  x,
  file,
  id = NA_character_,
  syntaxName = "rdfxml",
  mimeType = "application/rdf+xml",
 namespaces = data.frame(namespace = character(), prefix = character(), stringsAsFactors
    = FALSE),
  syntaxURI = NA_character_,
  resolveURI = NA_character_,
  creator = NA_character_
)
```

Arguments

x	A DataPackage object
...	Additional arguments
file	The file to which the ResourceMap will be serialized
id	A unique identifier for the serialization. The default value is the id assigned to the DataPackage when it was created.
syntaxName	The name of the syntax to use for serialization - default is "rdfxml"
mimeType	The mimetype of the serialized output - the default is "application/rdf+xml"
namespaces	A data frame containing one or more namespaces and their associated prefix
syntaxURI	URI of the serialization syntax
resolveURI	A character string containing a URI to prepend to datapackage identifiers
creator	A character string containing the creator of the package.

Details

The resource map that is created is serialized by default as RDF/XML. Other serialization formats can be specified using the syntaxName and mimeType parameters. Other available formats include:

syntaxName	mimeType
json	application/json
ntriples	application/n-triples
turtle	text/turtle
dot	text/x-graphviz

Note that the syntaxName and mimeType arguments together specify o serialization format.

Also, for packages that will be uploaded to the DataONE network, "rdfxml" is the only accepted format.

The resolveURI string value is prepended to DataPackage member identifiers in the resulting resource map. If no resolveURI value is specified, then 'https://cn.dataone.org/cn/v1/resolve' is used.

See Also

DataPackage-class

Examples

```
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
do <- new("DataObject", id="do1", dataobj=data, format="text/csv", user="jsmith")
dp <- addMember(dp, do)
data2 <- charToRaw("7,8,9\n10,11,12")
do2 <- new("DataObject", id="do2", dataobj=data2, format="text/csv", user="jsmith")
dp <- addMember(dp, do2)
dp <- describeWorkflow(dp, sources=do, derivations=do2)
## Not run:
```

```
td <- tempdir()
status <- serializePackage(dp, file=paste(td, "resmap.json", sep="/"), syntaxName="json",
    mimeType="application/json")
status <- serializePackage(dp, file=paste(td, "resmap.xml", sep="/"), syntaxName="rdfxml",
    mimeType="application/rdf+xml")
status <- serializePackage(dp, file=paste(td, "resmap.ttl", sep="/"), syntaxName="turtle",
    mimeType="text/turtle")

## End(Not run)
```

`serializeRDF` *Serialize a ResouceMap*

Description

The Redland RDF library is used to serialize the ResourceMap RDF model to a file as RDF/XML.

Usage

```
serializeRDF(x, ...)

## S4 method for signature 'ResourceMap'
serializeRDF(
  x,
  file,
  syntaxName = "rdfxml",
  mimeType = "application/rdf+xml",
 namespaces = data.frame(namespace = character(), prefix = character(), stringsAsFactors
    = FALSE),
  syntaxURI = NA_character_
)
```

Arguments

`x`	a ResourceMap
`...`	Additional parameters
`file`	the file to which the ResourceMap will be serialized
`syntaxName`	name of the syntax to use for serialization - default is "rdfxml"
`mimeType`	the mimetype of the serialized output - the default is "application/rdf+xml"
`namespaces`	a data frame containing one or more namespaces and their associated prefix
`syntaxURI`	A URI of the serialized syntax

Value

status of the serialization (non)

See Also

ResourceMap-class

Examples

```
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
do1 <- new("DataObject", id="id1", data, format="text/csv")
do2 <- new("DataObject", id="id2", data, format="text/csv")
dp <- addMember(dp, do1)
dp <- addMember(dp, do2)
dp <- insertRelationship(dp, subjectID="id1", objectIDs="id2",
  predicate="http://www.w3.org/ns/prov#wasDerivedFrom")
relations <- getRelationships(dp)
resmap <- new("ResourceMap")
resmap <- createFromTriples(resmap, relations, id="myuniqueid")
## Not run:
tf <- tempfile(fileext=".xml")
serializeRDF(resmap, tf)

## End(Not run)
```

---

serializeSystemMetadata

Serialize a SystemMetadata object to an XML representation

Description

The SystemMetadata object is converted to XML and written to a file.

Usage

```
serializeSystemMetadata(x, ...)

## S4 method for signature 'SystemMetadata'
serializeSystemMetadata(x, version = "v1", ...)
```

Arguments

x	The SystemMetadata instance to be serialized.
...	(Not currently used)
version	A character string representing the DataONE API version that this system will be used with (e.g. "v1", "v2").

Details

If the 'version' parameter is specified as *v2* then the SystemMetadata object is serialized according to the DataONE version 2.0 system metadata format.

Value

A character value of the filename that the XML representation of the SystemMetadata object was written to.

the character string representing a SystemMetadata object

See Also

SystemMetadata-class

Examples

```
library(XML)
doc <- xmlParseDoc(system.file("testfiles/sysmeta.xml", package="datapack"), asText=FALSE)
sysmeta <- new("SystemMetadata")
sysmeta <- parseSystemMetadata(sysmeta, xmlRoot(doc))
sysmetaXML <- serializeSystemMetadata(sysmeta, version="v2")
```

---

serializeToBagIt *Serialize A DataPackage into a BagIt Archive File*

Description

The BagIt packaging format https://datatracker.ietf.org/doc/html/draft-kunze-bagit-08 is used to prepare an archive file that contains the contents of a DataPackage.

Usage

```
serializeToBagIt(x, ...)

## S4 method for signature 'DataPackage'
serializeToBagIt(
  x,
  mapId = NA_character_,
  syntaxName = NA_character_,
  namespaces = data.frame(),
  mimeType = NA_character_,
  syntaxURI = NA_character_,
  resolveURI = NA_character_,
  creator = NA_character_,
  ...
)
```

Arguments

x	A DataPackage object
...	Additional arguments
mapId	A unique identifier for the package resource map. If not specified, one will be automatically generated.
syntaxName	The name of the syntax to use for the resource map serialization, defaults to "rdfxml"
namespaces	An optional data frame containing one or more namespaces and their associated prefix for the resource map serialization.
mimeType	The mimetype for the resource map serialization, defaults to "application/rdf+xml".
syntaxURI	An optional string specifying the URI for the resource map serialization.
resolveURI	A character string containing a URI to prepend to datapackage identifiers for the resource map.
creator	A character string containing the creator of the package.

Details

A BagIt Archive File is created by copying each member of a DataPackage, and preparing files that describe the files in the archive, including information about the size of the files and a checksum for each file. An OAI-ORE resource map is automatically created and added to the archive. These metadata files and the data files are then packaged into a single zip file.

Value

The file name that contains the BagIt zip archive. Recursively determines the name for a science metadata object. The base file name (eml, datacite, science-metadata, etc) should stay the same. Call the method with the base name and the number of existing files to start with. This is most likely 0. If there's a count defined, add it to the end of the file in () Then call the method again with count += 1 Eventually a free file name will be found, and then the function returns that name

See Also

DataPackage-class

For more information and examples regarding the parameters specifying the creation of the resource map, see serializePackage.

Examples

```
# Create the first data object
dp <- new("DataPackage")
data <- charToRaw("1,2,3,5,6")
do <- new("DataObject", id="do1", dataobj=data, format="text/csv", user="jsmith")
dp <- addMember(dp, do)
# Create a second data object
data2 <- charToRaw("7,8,9,4,10,11")
do2 <- new("DataObject", id="do2", dataobj=data2, format="text/csv", user="jsmith")
dp <- addMember(dp, do2)
```

```
# Create a relationship between the two data objects
dp <- describeWorkflow(dp, sources="do2", derivations="do2")
# Write out the data package to a BagIt file
## Not run:
bagitFile <- serializeToBagIt(dp, syntaxName="json", mimeType="application/json")

## End(Not run)
```

`setPublicAccess` *Add a Rule to the AccessPolicy to make the object publicly readable*

Description

To be called prior to creating the object in DataONE. When called before creating the object, adds a rule to the access policy that makes this object publicly readable. If called after creation, it will only change the system metadata locally, and will not have any effect on remotely uploaded copies of the DataObject.

Usage

```
setPublicAccess(x, ...)

## S4 method for signature 'DataObject'
setPublicAccess(x)

## S4 method for signature 'DataPackage'
setPublicAccess(x, identifiers = list())
```

Arguments

`x`	DataObject
`...`	(not yet used)
`identifiers`	A list of `character` values containing package member identifiers that will be updated (default is all package members).

Value

A DataObject with modified access rules.

A DataPackage with modified access rules.

See Also

`DataObject-class`

`DataObject-class`

`DataPackage-class`

Examples

```
data <- charToRaw("1,2,3\n4,5,6\n")
do <- new("DataObject", "id1", dataobj=data, "text/csv",
  "uid=jones,DC=example,DC=com", "urn:node:KNB")
do <- setPublicAccess(do)
# First create a sample package with two DataObjects
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6\n")
obj <- new("DataObject", id="id1", dataobj=data, format="text/csv")
dp <- addMember(dp, obj)
data2 <- charToRaw("7,8,9\n4,10,11\n")
obj2 <- new("DataObject", id="id2", dataobj=data2, format="text/csv")
dp <- addMember(dp, obj2)
# Now add public read to all package members ("id1", "id2")
dp <- setPublicAccess(dp)
```

---

setValue *Set values for selected DataPackage members*

Description

The 'setValue' method is used to modify values stored in DataPackage members. Each member in a DataPackage is a DataObject which is an R S4 object that contains a set of values (slots). The available slots are described at help("DataObject-class").

Usage

```
setValue(x, ...)

## S4 method for signature 'DataPackage'
setValue(x, name, value, identifiers = NA_character_, ...)
```

Arguments

x	A DataPackage instance
...	(Not yet used)
name	A DataObject slot name.
value	A new value to assign to the slot for selected DataPackage members.
identifiers	A list of identifiers of DataPackage members to update.

Details

If the parameter identifiers is provided, then DataPackage members that have identifiers specified in the list will be updated. If this parameter is not provided then no members will be updated. To update all members in a package, specify the value of identifiers=getIdentifiers(pkg) where pkg is the variable name of the DataPackage to update. Note that this method can be used to update the data or filenane slots, but it is instead recommended to us the replaceMember method to achieve this, as the replaceMember method assists in properly setting the related SystemMetadata values.

Value

A DataPackage with possibly updated DataObjects.

See Also

DataPackage-class

Examples

```
# First create a package that we can modify.
dp <- new("DataPackage")
data <- charToRaw("1,2,3\n4,5,6")
# The next statment sets the format type incorrectly as an example, so we can correct it later
do <- new("DataObject", id="myNewId", dataobj=data, format="image/jpg", user="jsmith")
dp <- addMember(dp, do)
data <- charToRaw("7,8.9\n4,10,11")
# This next statement also sets the format type incorrectly
do <- new("DataObject", id="myNewId2", dataobj=data, format="image/jpg", user="jsmith")
dp <- addMember(dp, do)
# Change format types to correct value for both package members
# Careful! Specifying 'identifiers=getIdentifiers(dp) will update all package members!
dp <- setValue(dp, name="sysmeta@formatId", value="text/csv", identifiers=getIdentifiers(dp))
```

SystemMetadata *Create DataONE SystemMetadata object*

Description

A class representing DataONE SystemMetadata, which is core information about objects stored in a repository and needed to manage those objects across systems. SystemMetadata contains basic identification, ownership, access policy, replication policy, and related metadata.

If the *sysmeta* parameter is specified, then construct a new SystemMetadata instance by using the fields from an XML representation of the SystemMetadata.

Usage

```
SystemMetadata(...)

## S4 method for signature 'XMLInternalElementNode'
SystemMetadata(x, ...)
```

Arguments

...	Additional arguments
x	A value of type "XMLInternalElementNode", containing the parsed XML element with SystemMetadata fields.

See Also

SystemMetadata-class

SystemMetadata-class *A DataONE SystemMetadata object containing basic identification, ownership, access policy, replication policy, and related metadata*

Description

A class representing DataONE SystemMetadata, which is core information about objects stored in a repository and needed to manage those objects across systems. SystemMetadata contains basic identification, ownership, access policy, replication policy, and related metadata.

Slots

`serialVersion` value of type `"numeric"`, the current version of this system metadata; only update the current version

`identifier` value of type `"character"`, the identifier of the object that this system metadata describes.

`replicationAllowed` value of type `"logical"`, replication policy allows replicas.

`numberReplicas` value of type `"numeric"`, for number of supported replicas.

`preferredNodes` value of type `"list"`, of preferred member nodes.

`blockedNodes` value of type `"list"`, of blocked member nodes.

`formatId` value of type `"character"`, the DataONE object format for the object.

`size` value of type `"numeric"`, the size of the object in bytes.

`checksum` value of type `"character"`, the checksum for the object using the designated checksum algorithm.

`checksumAlgorithm` value of type `"character"`, the name of the hash function used to generate a checksum, from the DataONE controlled list.

`submitter` value of type `"character"`, the Distinguished Name or identifier of the person submitting the object.

`rightsHolder` value of type `"character"`, the Distinguished Name or identifier of the person who holds access rights to the object.

`accessPolicy` value of type `"data.frame"`, a list of access rules as (subject, permission) tuples to be applied to the object.

`obsoletes` value of type `"character"`, the identifier of an object which this object replaces.

`obsoletedBy` value of type `"character"`, the identifier of an object that replaces this object.

`archived` value of type `"logical"`, a boolean flag indicating whether the object has been archived and thus hidden.

`dateUploaded` value of type `"character"`, the date on which the object was uploaded to a member node.

`dateSysMetadataModified` value of type `"character"`, the last date on which this system metadata was modified.

`originMemberNode` value of type `"character"`, the node identifier of the node on which the object was originally registered.

`authoritativeMemberNode` value of type `"character"`, the node identifier of the node which currently is authoritative for the object.

`seriesId` value of type `"character"`, a unique Unicode string that identifies an object revision chain. A seriesId will resolve to the latest version of an object.

`mediaType` value of type `"character"`, the IANA Media Type (aka MIME-Type) of the object, e.g. "text/csv".

`fileName` value of type `"character"`, the name of the file to create when this object is downloaded from DataONE.

`mediaTypeProperty` value of type a `"list"` of `"character"`, IANA Media Type properties for the `"mediaType"` argument

Methods

`initialize` : Initialize a DataONE SystemMetadata object with default values or values passed in to the constructor object

`SystemMetadata` : Create a SystemMetadata object, with all fields set to the value found in an XML document

`parseSystemMetadata` : Parse an external XML document and populate a SystemMetadata object with the parsed data

`serializeSystemMetadata` : Get the Count of Objects in the Package

`validate` : Validate a SystemMetadata object

`addAccessRule` : Add access rules to an object such as system metadata

`hasAccessRule` : Determine if a particular access rules exists within SystemMetadata.

`clearAccessPolicy` : Clear the accessPolicy from the specified object.

See Also

`datapack`

`updateMetadata` *Update selected elements of the XML content of a DataObject in a DataPackage (aka package member)*

Description

A DataObject that contains an XML document can be edited by specifying a path to the elements to edit (an XPath expression) and a value to replace the text node.

Usage

```
updateMetadata(x, do, ...)

## S4 method for signature 'DataPackage'
updateMetadata(x, do, xpath, replacement, newId = NA_character_, ...)
```

Arguments

x	a DataPackage instance
do	A DataObject instance object, or DataObject identifier
...	(Not yet used)
xpath	A character value specifying the location in the XML to update.
replacement	A character value that will replace the elements found with the xpath.
newId	A value of type "character" which will replace the identifier for this DataObject.

Details

This method requires some knowledge of the structure of the metadata document as well as facility with the XPath language. If the newId argument is used, the specified new identifier will be assigned to the object, and the previous identifier will be stored in the oldId slot, for possible use when updating the DataObject to a repository. If newId is not used, a new identifier will be generated for the DataObject only the first time that updateMetadata is called for a particular object in a DataPackage.

See Also

DataPackage-class

Examples

```
# Create a DataObject and add it to the DataPackage
dp <- new("DataPackage")
sampleMeta <- system.file("./extdata/sample-eml.xml", package="datapack")
id <- "1234"
metaObj <- new("DataObject", id="1234", format="eml://ecoinformatics.org/eml-2.1.1",
               file=sampleMeta)
dp <- addMember(dp, metaObj)

# In the metadata object, insert the newly assigned data
xp <- sprintf("//dataTable/physical/distribution[../objectName/text()=\"%s\"]/online/url",
              "sample-data.csv")
newURL <- sprintf("https://cn.dataone.org/cn/v2/resolve/%s", "1234")
dp <- updateMetadata(dp, id, xpath=xp, replacement=newURL)
```

updateRelationships *Update package relationships by replacing an old identifier with a new one*

Description

When package members are updated, they receive a new identifier (replaceMember). It is therefor necessary to update the package relationships to update occurrences of the old identifier with the new one when the old identifier appears in the "subject" or "object" of a relationship.

Usage

```
updateRelationships(x, ...)

## S4 method for signature 'DataPackage'
updateRelationships(x, id, newId, ...)
```

Arguments

x	A DataPackage object
...	(Not yet used)
id	A character value containing the identifier to be replaced.
newId	A character value containing the identifier that will replace the old identifier.

See Also

DataPackage-class

updateXML *Update selected elements of the XML content of a DataObject*

Description

The data content of the DataObject is updated by using the xpath argument to locate the elements to update with the character value specified in the replacement argument.

Usage

```
updateXML(x, ...)

## S4 method for signature 'DataObject'
updateXML(x, xpath = NA_character_, replacement = NA_character_, ...)
```

Arguments

x	A DataObject instance
...	Additional parameters (not yet used)
xpath	A character value specifying the location in the XML to update.
replacement	A character value that will replace the elements found with the xpath.

Value

The modified DataObject

See Also

DataObject-class

Examples

```
## Not run:
library(datapack)
dataObj <- new("DataObject", format="text/csv", file=sampleData)
sampleEML <- system.file("extdata/sample-eml.xml", package="datapack")
dataObj <- updateMetadata(dataObj, xpath="", replacement=)

## End(Not run)
library(datapack)
# Create the metadata object with a sample EML file
sampleMeta <- system.file("./extdata/sample-eml.xml", package="datapack")
metaObj <- new("DataObject", format="eml://ecoinformatics.org/eml-2.1.1", file=sampleMeta)
# In the metadata object, replace "sample-data.csv" with 'sample-data.csv.zip'
xp <- sprintf("//dataTable/physical/objectName[text()=\"%s\"]", "sample-data.csv")
metaObj <- updateXML(metaObj, xpath=xp, replacement="sample-data.csv.zip")
```

---

validate *Validate a SystemMetadata object*

Description

Validate a system metadata object, ensuring that required fields are present and of the right type.

Usage

```
validate(x, ...)

## S4 method for signature 'SystemMetadata'
validate(x, ...)
```

Arguments

x	the instance to be validated
...	(Additional parameters)

Value

logical, TRUE if the SystemMetadata object is valid, else a list of strings detailing errors

See Also

SystemMetadata-class

Examples

```
library(XML)
doc <- xmlParseDoc(system.file("testfiles/sysmeta.xml", package="datapack"), asText=FALSE)
sysmeta <- new("SystemMetadata")
sysmeta <- parseSystemMetadata(sysmeta, xmlRoot(doc))
valid <- validate(sysmeta)
```

Index

∗ **classes**
DataObject-class, 12
∗ **resourceMap**
ResourceMap-class, 44

addAccessRule, 3, *13*, *15*, *56*
addAccessRule,DataObject-method *(addAccessRule)*, 3
addAccessRule,DataPackage-method *(addAccessRule)*, 3
addAccessRule,SystemMetadata-method *(addAccessRule)*, 3
addData, 5
addData,DataPackage,DataObject-method *(addData)*, 5
addMember, 6, *15*
addMember,DataPackage-method *(addMember)*, 6

calculateChecksum, 7
calculateChecksum,DataObject-method *(calculateChecksum)*, 7
canRead, 8, *13*
canRead,DataObject-method *(canRead)*, 8
clearAccessPolicy, 9, *15*, *56*
clearAccessPolicy,DataObject-method *(clearAccessPolicy)*, 9
clearAccessPolicy,DataPackage-method *(clearAccessPolicy)*, 9
clearAccessPolicy,SystemMetadata-method *(clearAccessPolicy)*, 9
containsId, 10, *15*
containsId,DataPackage-method *(containsId)*, 10
createFromTriples, 11, *44*
createFromTriples,ResourceMap-method *(createFromTriples)*, 11

DataObject-class, 12
DataObject-initialize *(initialize,DataObject-method)*, 27
datapack, *13*, *15*, *44*, *56*
DataPackage-class, 14
DataPackage-initialize *(initialize,DataPackage-method)*, 29
describeWorkflow, *15*, 16
describeWorkflow,DataPackage-method *(describeWorkflow)*, 16
dmsg, 17

freeResourceMap, 18
freeResourceMap,ResourceMap-method *(freeResourceMap)*, 18

getData, *13*, *15*, 18
getData,DataObject-method *(getData)*, 18
getData,DataPackage-method *(getData)*, 18
getFormatId, *13*, 19
getFormatId,DataObject-method *(getFormatId)*, 19
getIdentifier, *13*, 20
getIdentifier,DataObject-method *(getIdentifier)*, 20
getIdentifiers, *15*, 21
getIdentifiers,DataPackage-method *(getIdentifiers)*, 21
getMember, *15*, 21
getMember,DataPackage-method *(getMember)*, 21
getRelationships, *11*, *15*, 22
getRelationships,DataPackage-method *(getRelationships)*, 22
getSize, *15*, 23
getSize,DataPackage-method *(getSize)*, 23
getTriples, 24, *44*
getTriples,ResourceMap-method *(getTriples)*, 24

getValue, *15*, 25
getValue,DataPackage-method *(*getValue*)*, 25

hasAccessRule, *13*, *15*, 26, *56*
hasAccessRule,DataObject-method *(*hasAccessRule*)*, 26
hasAccessRule,DataPackage-method *(*hasAccessRule*)*, 26
hasAccessRule,SystemMetadata-method *(*hasAccessRule*)*, 26

initialize, *13*, *15*, *44*, *56*
initialize,DataObject-method, 27
initialize,DataPackage-method, 29
initialize,ResourceMap-method, 30
initialize,SystemMetadata-method, 30
insertRelationship, *15*, 33
insertRelationship,DataPackage-method *(*insertRelationship*)*, 33

parseRDF, 34, *44*
parseRDF,ResourceMap-method *(*parseRDF*)*, 34
parseSystemMetadata, 35, *56*
parseSystemMetadata,SystemMetadata-method *(*parseSystemMetadata*)*, 35
plotRelationships, 36
plotRelationships,DataPackage-method *(*plotRelationships*)*, 36

recordDerivation, 37
recordDerivation,DataPackage-method *(*recordDerivation*)*, 37
removeAccessRule, *15*, 38
removeAccessRule,DataObject-method *(*removeAccessRule*)*, 38
removeAccessRule,DataPackage-method *(*removeAccessRule*)*, 38
removeAccessRule,SystemMetadata-method *(*removeAccessRule*)*, 38
removeMember, *15*, 40
removeMember,DataPackage-method *(*removeMember*)*, 40
removeRelationships, *15*, 41
removeRelationships,DataPackage-method *(*removeRelationships*)*, 41
replaceMember, *15*, 42
replaceMember,DataPackage-method *(*replaceMember*)*, 42

ResourceMap-class, 44
ResourceMap-initialize *(*initialize,ResourceMap-method*)*, 30

selectMember, *15*, 45
selectMember,DataPackage-method *(*selectMember*)*, 45
serializePackage, *15*, 46, *51*
serializePackage,DataPackage-method *(*serializePackage*)*, 46
serializeRDF, *44*, 48
serializeRDF,ResourceMap-method *(*serializeRDF*)*, 48
serializeSystemMetadata, 49, *56*
serializeSystemMetadata,SystemMetadata-method *(*serializeSystemMetadata*)*, 49
serializeToBagIt, *15*, 50
serializeToBagIt,DataPackage-method *(*serializeToBagIt*)*, 50
setPublicAccess, *13*, *15*, 52
setPublicAccess,DataObject-method *(*setPublicAccess*)*, 52
setPublicAccess,DataPackage-method *(*setPublicAccess*)*, 52
setValue, *15*, 53
setValue,DataPackage-method *(*setValue*)*, 53
show, *15*
SystemMetadata, *12*, 54, *56*
SystemMetadata,XMLInternalElementNode-method *(*SystemMetadata*)*, 54
SystemMetadata-class, 55
SystemMetadata-initialize *(*initialize,SystemMetadata-method*)*, 30

updateMetadata, *15*, 56
updateMetadata,DataPackage-method *(*updateMetadata*)*, 56
updateRelationships, *15*, 58
updateRelationships,DataPackage-method *(*updateRelationships*)*, 58
updateXML, *13*, 58
updateXML,DataObject-method *(*updateXML*)*, 58

validate, *56*, 59

validate,SystemMetadata-method
(validate), 59