Package 'datapack'

April 8, 2017

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

Version 1.2.0

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://tools.ietf.org/html/draft-kunze-bagit-08>.

Date 2017-04-07

Depends R (>= 3.1.1)

Imports digest, methods, redland, XML, hash, uuid

Suggests testthat, knitr

License Apache License (== 2.0)

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

LazyData true

VignetteBuilder knitr

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

RoxygenNote 6.0.1

NeedsCompilation no

Author Matthew B. Jones [aut, cre], Peter Slaughter [aut], Regents of the University of California [cph]

Maintainer Matthew B. Jones <jones@nceas.ucsb.edu>

Repository CRAN

Date/Publication 2017-04-07 22:59:01 UTC

R topics documented:

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, ...)
```

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

Value

the SystemMetadata object with the updated access policy.

the DataObject with the updated access policy

See Also

SystemMetadata-class

Examples

```
# 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)
```

```
data <- charToRaw("1,2,3\n4,5,6\n")
obj <- new("DataObject", id="1234", data=data, format="text/csv")
obj <- addAccessRule(obj, "uid=smith,ou=Account,dc=example,dc=com", "write")
```

---

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 = as.character(NA))
```

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.
dpkg <- addData(dpkg, do, md)
```

---

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 enfirced 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", data=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, ...)
```

Arguments

x	the instance to clear access rules from.
...	(Additional parameters)

Value

The SystemMetadata object with the cleared access policy.

See Also

SystemMetadata-class

Examples

```
sysmeta <- new("SystemMetadata")
sysmeta <- addAccessRule(sysmeta, "uid=smith,ou=Account,dc=example,dc=com", "write")
sysmeta <- clearAccessPolicy(sysmeta)
```

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 <- addData(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 = as.character(NA), externalIdentifiers = list(), ...)
```

Arguments

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

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 <- addData(dp, do1)
dp <- addData(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

Methods

- initialize: Initialize a DataObject
- getData: Get the data content of a specified data object
- getIdentifier: Get the Identifier of the DataObject
- getFormatId: Get the FormatId of the DataObject
- setPublicAccess: Add a Rule to the AccessPolicy to make the object publicly readable.
- addAccessRule: Add a Rule to the AccessPolicy
- canRead: Test whether the provided subject can read an object.

See Also

datapack

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")
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)
do <- new("DataObject", "id1", format="text/csv", user="uid=jones,DC=example,DC=com",
  mnNodeId="urn:node:KNB", filename=tf)

## End(Not run)
```

---

datapack *datapack, a container for packages of data and associated metadata*

Description

The datapack R package provides an abstraction for collating heterogeneous collections of data objects and metadata into a bundle that can be transported and loaded into a single composite file. The methods in this package provide a convenient way to load data from common repositories such as DataONE into the R environment, and to document, serialize, and save data from R to data repositories workdwide. A data package is represented as an instance of the S4 class DataPackage,

which consists of one or more instances of the S4 DataObject class, which in turn contains an instance of the S4 SystemMetadata class. The SystemMetadata class provides critical metadata about a data object that is needed to transport it to an external repository, including the identifier for the object, its format, its checksum and size, and information about which repositories the data is associated with. DataPackages can be loaded from and saved to the DataONE federated network of repositories using the dataone package, but they can also be used as standalone transport containers for other systems.

A DataPackage includes a manifest based on the OAI-ORE specification for describing aggregations of files as a ResourceMap. Resource maps are RDF documents that conform to the Open Archives Initiative Object Reuse and Exchange (OAI-ORE) specification. Resource maps are generated by data providers to define data packages, and have a namespace of http://www.openarchives.org/ore/terms/.

A DataPackage is serialized as a zip file following the BagIt RFC specification, which provides a consistent mechanism for a serialized representation of a group of opaque objects in a predictable structure. BagIt includes a specification for including metadata about each of the objects, the bag itself, and fixity attributes so that any BagIt implementation can validate the components contained within a package. When expanded, a BagIt zipfile will expand to a common directory structure with a predictable set of metadata that describes the structure and content of the bag. Conformance with the BagIt specification is handled by the DataPackage class.

Classes

- `DataPackage-class`: A class representing a data package, which can contain data objects
- `DataObject-class`: DataObject wraps raw data with system-level metadata
- `SystemMetadata-class{SystemMetadata}`: A DataONE SystemMetadata object containing basic identification, ownership, access policy, replication policy, and related metadata.
- `ResourceMap-class{ResourceMap}`: ResourceMap provides methods to create, serialize and deserialize an OAI ORE resource map.

Author(s)

Matthew B. Jones (NCEAS), Peter Slaughter (NCEAS)

`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 hash containing provenance relationships of package objects

objects A hash 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

Methods

- initialize: Initialize a DataPackage object
- 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
- addData: Add a DataObject to the DataPackage
- insertRelationship: Insert relationships between objects in a DataPackage
- getRelationships: Retrieve relationships of data package objects
- containsId: Returns true if the specified object is a member of the data package
- removeMember: Remove the Specified Member from the DataPackage
- getMember: Return the DataPackage Member by Identifier
- serializePackage: Create an OAI-ORE resource map from the data package
- serializeToBagIt: Serialize A DataPackage into a BagIt Archive File
- describeWorkflow: Add data derivation information to a DataPackage

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 = as.character(NA), derivations = list(), ...)
```

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.

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 <- addData(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 <- addData(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 <- addData(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
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 <- addData(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 <- addData(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 <- addData(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 ralationships 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 <- addData(dp, do)
getSize(dp)
```

---

hasAccessRule *Determine if a particular access rules exists within SystemMetadata.*

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.

Usage

```
hasAccessRule(x, ...)

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

Arguments

x	the SystemMetadata instance to which to add the rules
...	Additional arguments
subject	the subject of the rule to be checked
permission	the permission to be applied to subject if x is character

Value

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

boolean TRUE if the access rule exists already, FALSE otherwise

See Also

SystemMetadata-class

Examples

```
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")
```

`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 = as.character(NA),
  dataobj = NA, format = as.character(NA), user = as.character(NA),
  mnNodeId = as.character(NA), filename = as.character(NA),
  seriesId = as.character(NA), mediaType = as.character(NA),
  suggestedFilename = as.character(NA), mediaTypeProperty = list())
```

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).
`suggestedFilename`	A suggested filename to use when this object is serialized. If not specified, defaults to the basename of the filename parameter.
`mediaTypeProperty`	A list, indicates IANA Media Type properties to be associated with the parameter `"mediaType"`

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.

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")
```

---

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 = as.character(NA))
```

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 = as.character(NA),
  formatId = as.character(NA), size = as.numeric(NA),
  checksum = as.character(NA), checksumAlgorithm = "SHA-1",
  submitter = as.character(NA), rightsHolder = as.character(NA),
  accessPolicy = data.frame(subject = character(), permission = character()),
  replicationAllowed = TRUE, numberReplicas = 3,
  obsoletes = as.character(NA), obsoletedBy = as.character(NA),
  archived = FALSE, dateUploaded = as.character(NA),
  dateSysMetadataModified = as.character(NA),
  originMemberNode = as.character(NA),
  authoritativeMemberNode = as.character(NA), preferredNodes = list(),
  blockedNodes = list(), seriesId = as.character(NA),
  mediaType = as.character(NA), fileName = as.character(NA),
  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 replicants.
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"`, a suggested file name for the object (if the object containing this sysmeta is serialized).

`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 = as.character(NA), subjectType = as.character(NA),
  objectTypes = as.character(NA), dataTypeURIs = as.character(NA))
```

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 specifiy 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",
    "http://www.w3.org/ns/prov#used",
   "https://knb.ecoinformatics.org/knb/d1/mn/v1/object/doi:1234/_030MXTI009R00_20030812.40.1")
# Create a relationshp with the subject as a blank node with an automatically assigned blank node id
dp <- insertRelationship(dp, subjectID=as.character(NA), 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="_:BL1234", objectIDs="thing7",
    predicate="http://www.myns.org/hadThing")
# Create multiple relationships with the same subject, predicate, but different objects
```

```
dp <- insertRelationship(dp, subjectID="_:bl2", 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")
```

`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))
```

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 specifiy 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)
```

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

Description

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

Usage

```
removeMember(x, ...)

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

Arguments

`x`	a Datapackage object
`...`	(Not yet used)
`identifier`	an identifier for a DataObject

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 <- addData(dp, do)
removeMember(dp, "myNewId")
```

`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 aggregrations 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`: Get the data content of a specified data object
- `serializeRDF`: Get the Count of Objects in the Package

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)
```

`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 = as.character(NA),
  syntaxName = "rdfxml", mimeType = "application/rdf+xml",
  namespaces = data.frame(namespace = character(), prefix = character(),
  stringsAsFactors = FALSE), syntaxURI = as.character(NA),
  resolveURI = as.character(NA))
```

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 assinged 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

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 <- addData(dp, do)
data2 <- charToRaw("7,8,9\n10,11,12")
do2 <- new("DataObject", id="do2", dataobj=data2, format="text/csv", user="jsmith")
dp <- addData(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 = as.character(NA))
```

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 <- addData(dp, do1)
dp <- addData(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 (eg. "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 metdata 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://tools.ietf.org/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 = as.character(NA),
  syntaxName = as.character(NA), namespaces = data.frame(),
  mimeType = as.character(NA), syntaxURI = as.character(NA),
  resolveURI = as.character(NA), ...)
```

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 identifiersa for the resource map.

Details

A BagIt Archive File is created by copying each member of a DataPackge, 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.

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\n4,5,6")
do <- new("DataObject", id="do1", dataobj=data, format="text/csv", user="jsmith")
dp <- addData(dp, do)
# Create a second data object
data2 <- charToRaw("7,8,9\n4,10,11")
do2 <- new("DataObject", id="do2", dataobj=data2, format="text/csv", user="jsmith")
dp <- addData(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)
```

Arguments

x	DataObject
...	(not yet used)

Value

DataObject with modified access rules

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")
do <- setPublicAccess(do)
```

`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"`, for replication policy allows replicants.

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

`preferredNodes` value of type `"list"`, of prefered 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"`, a suggested file name for the object.

`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

`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