Package 'dataone'

February 24, 2013

Version 1.0.0

Date 2013-02-14

Title DataONE R Client

Author Matthew Jones, Rob Nahf

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

Description A package that provides read/write access to data and metadata from the DataONE network of Member Node data repositories.

Depends R (>= 2.9.2), rJava (>= 0.8-5), XML (>= 3.95-0.1), methods,dataonelibs

Imports rJava, XML

SystemRequirements Java (>= 1.6)

License Apache License 2.0

Repository CRAN

Date/Publication 2013-02-14 20:49:45

URL http://releases.dataone.org/online/dataone_r

BugReports https://redmine.dataone.org/projects/dataone_r

R topics documented:

dataone-package *DataONE R Client*

Description

A package that provides read/write access to data and metadata from the DataONE network of Member Node data repositories.

Details

Package:	dataone
Version:	1.0.0
Date:	2013-02-14
Depends:	R (>= 2.9.2), rJava (>= 0.8-5), XML (>= 3.95-0.1), methods
Imports:	rJava, XML
SystemRequirements:	Java (>= 1.6)
License:	file LICENSE
Packaged:	2013-02-14 20:54:27 UTC; jones
Repository:	CRAN
Date/Publication:	2013-02-14 20:49:45
URL:	http://www.dataone.org/investigator-toolkit/dataone_r
BugReports:	https://redmine.dataone.org
Built:	R 2.15.1; ; 2013-02-15 21:42:52 UTC; unix

Index:

```
AbstractTableDescriber-class
                        Class '"AbstractTableDescriber"'
CertificateManager-class
                        Class '"CertificateManager"'
D1Client-class          Class '"D1Client"'
D1Object-class          Class '"D1Object"'
DataPackage-class       Class '"DataPackage"'
EMLParser-class         Class '"EMLParser"'
dataone-package         Package to access data and metadata in the
                        DataONE repository federation.
```

Getting Started

The R Client addresses 5 broad functional areas: client setup, search, data retrieval, data submission, and dataFrame interoperability.

The dataone package uses S4 classes and methods, so finding help for each of the dataone classes can be accomplished with the command:

```
> class ? <theClass>
```

for example:

```
> class ? D1Client
```

1. Client Setup: Client setup includes setting your member node (where submissions will go), and setting up your client subject. (You are identified to DataONE by your client subject). Most interaction with the DataONE system is mediated by the D1Client class - retrievals, searches, submissions. The D1Client 'constructor' method builds a D1Client object configured to the chosen environment and membernode.

some examples:

```
> cli <- D1Client() # builds a client to the production environment

> cli <- D1Client(,"urn:node:WERSDF")   # builds a client to the
                                        # production environment and sets
                                        # the default member node

> cli <- D1Client("DEV", "urn:node:UIYOP")  # sets the environment to DEV,
                                            # and sets the default member node
```

There are also some helper functions for managing your client subject.

```
> cm <- CertificateManager()
> downloadCert(cm)    # opens the CILogon page in your default browser,
                      # to assist in getting your client certificate
                      # downloaded
> getCertExpires(cm)  # displays the date-time that your current
                      # certificate is valid until.
```

For documentation on all of the CertificateManager helper functions, type:

```
> class ? CertificateManager
```

2. Data Search: DataONE coordinating nodes expose a SOLR query endpoint that can be queried against to get information about stored objects. Those familiar with SOLR queries can use the D1Client methods `d1SolrQuery`

```
> results <- d1SolrQuery(cli, list(q="foo",fl="identifier,etc..."))
```

to return solr results for their own interpretation.

For more streamlined searches to get just the identifiers:

```
> d1IdentifierSearch(cli, "q=foo")
```

returns a character vector of the identifiers of records found using the word 'foo' as a search term.

Future development based on initial feedback on these search methods is planned.

3. Data Retrieval: Data retrieval from DataONE is accomplished using an object Identifier, obtained either from a data search within R, from ONEMercury, or even in a published paper.

Ideally, the data you want to retrieve has been submitted as part of a package, and you will be retrieving the entire package.

A package in DataONE terminology is a set of individual objects (files, usually) that relate to each other, and whose relationships to each other are defined in a separate object called a Resource Map. To retrieve a data package, use the following:

```
> cli <- D1Client()
> packageOfInterest <- getPackage(cli,"interestingStuff")
```

the object 'packageOfInterest' is of type "DataPackage", and once you have it, you can start looking at the individual objects that make it up:

```
> members <- getIdentifiers(packageOfInterest)
> member1 <- getMember(packageOfInterest,members[1])
> member1[0]
"D1Object"
```

At this point, you will notice that the individual objects are represented locally with the objects of class "D1Object". What's in it?

```
> getData(member1)  # returns the content of member1
```

At times you may wish to retrieve an individual object directly, without retrieving the entire package. Use the following method, `getD1Object`.

```
> cli <- D1Client()
> item <- getD1Object(cli,"interestingObject")
```

A list of methods used for working with DataPackages and their member D1Objects can be found with:

```
> methods ? DataPackage
> methods ? D1Object
```

4. Submitting Data:

Data submission functionality is still in development. We have released a basic set of functions for the three major related activities: assembling the package, attending to who will have access to it, and submitting to DataONE. However, the functionality for the second two activities is limited to setting public access to objects, and creating new data. Future releases will support content updates, archiving, and fine tuning access policies.

The best practice is to submit new data as part of a package containing the data, the metadata that describes it, and the ORE resource map that defines the relationship between the two (or more). Typically, the scope of a package is 1 metadata object along with 1 or more data objects it documents. DataONE does support packages with multiple metadata objects and their data as well.

The DataPackage class provides methods for assembling the data and metadata objects and defining the "documents / documented-by" relationships that get fed into the resource map. All that is needed do is adding the members of the data package, and telling it which (metadata) members document which (data) members. After that, you submit the dataPackage. For example:

```
env.label <- "STAGING"
mn.nodeid <- "urn:node:foo"
d1.client <- D1Client(env.label, mn.nodeid)

d1o.d1 <- new("D1Object", id.d1, table.1.data, data.formatID, mn.nodeid)
d1o.d2 <- new("D1Object", id.d2, table.2.data, data.formatID, mn.nodeid)
d1o.d3 <- new("D1Object", id.d3, table.3.data, data.formatID, mn.nodeid)
d1o.md1 <- new("D1Object", id.md1, metadata, md.formatID, mn.nodeid)

## let's make these publicly readable
setPublicAccess(d1o.d1)
setPublicAccess(d1o.d2)
setPublicAccess(d1o.d3)
setPublicAccess(d1o.md1)

data.package <- new("DataPackage",packageId=packageId)

addData(data.package,d1o.d1)
```

```
addData(data.package,d1o.d2)
addData(data.package,d1o.d3)
addData(data.package,d1o.md1)
insertRelationship(data.package, id.md1, c(id.d1, id.d2, id.d3))

create(d1.client, data.package)
```

You might have noticed that there is no mechanism to make the resourceMap itself a public object. Currently, the DataPackage create method makes all of the resourceMaps it creates public (after it creates the resourceMap.) This definitely needs to change.

5. dataFrame Interoperability: Once you have access to dataone content in R, it would be nice to have that data in a useful R data structure, specifically the ubiquitous dataFrame. This package provides functions to serialize dataFrames (to .csv), and convert them back into dataFrames. Serializing as csv makes the dataset almost universally useful for others.
A simple example:

```
## serialize to csv
## you can run this one!
library(dataone)
## let's load in a built-in dataset that's a dataFrame
data(trees)
cli <- D1Client("SANDBOX")
serializedTreeData <- convert.csv(cli, trees)
dataFormatId <- "text/csv"

treeData <- new("D1Object","treesId",
                serializedTreeData, dataFormatId, "urn:node:mnDemo5")

## show the serialized form of the trees data
getData(treeData)

## should display the original trees dataFrame
asDataFrame(treeData)
```

Some metadata contains information on how the data is serialized and stored, and can provide useful information on how to deserialize the stored content correctly. The power of working with data packages, instead of the individual object is revealed by this.
The dataone R client provides additional `asDataFrame` methods to allow the associated metadata that contains content storage information to participate in deserialization.
examples:

```
df <- asDataFrame(data.package, dataMember1.id)

df <- asDataFrame(data.object, its.metadata)

table.describer <- EMLParser(its.metadata)
df <- asDataFrame(data.object, table.describer)

df <- asDataFrame(data.object, sep="\t", ...)

df <- asDataFrame(data.object)
```

In the first example the dataPackage uses the resource map to associate the data object with its describing metadata, and send that along to the deserializer. The second example is useful where there's no data package to do the association, but it is known by some other means.

Currently, the dataone package can only parse EML documents, but has the ability to plug in other metadata format parsers as they become available, by way of the AbstractTableDescriber virtual class.

Author(s)

Matthew Jones, Rob Nahf

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

AbstractTableDescriber-class

Class "AbstractTableDescriber"

Description

A virtual class (interface) that defines the methods used by the corresponding asDataFrame method to get parsing instructions.

Classes that inherit from this class provide the format-specific ways to read the metadata.

Objects from the Class

A virtual Class: No objects may be created from it.

Methods

asDataFrame `signature(x = "D1Object", reference = "AbstractTableDescriber")`: ...

Generics

data.characterEncoding `signature(x = "AbstractTableDescriber", index = "numeric")`: return the character encoding entry from the metadata

data.formatFamily `signature(x = "AbstractTableDescriber", index = "numeric")`: return the formatFamily of the data object being described. asDataFrame methods only work on 'text/simpleDelimited'

data.tableAttributeNames `signature(x = "AbstractTableDescriber", index = "numeric")`: returns the attribute names defined in the metadata document for the specified data table

data.tableAttributeOrientation `signature(x = "AbstractTableDescriber", index = "numeric")`: returns either "rows" or "columns", based on which way the attribute headers run. Most data has a header row where the attribute names go across "columns", in which case, the return value for this method should be "columns."

data.tableAttributeStorageTypes `signature(x = "AbstractTableDescriber", index = "numeric")`: returns the attributes' data storage types defined in the metadata document for the specified data table

data.tableAttributeTypes `signature(x = "AbstractTableDescriber", index = "numeric")`: returns the attributes' data types defined in the metadata document for the specified data table

data.tableFieldDelimiter `signature(x = "AbstractTableDescriber", index = "numeric")`: the field delimiter(s) of the data object being described

data.tableMissingValueCodes `signature(x = "AbstractTableDescriber", index = "numeric")`: returns the missing value codes defined in the metadata document for the specified data table

data.tableQuoteCharacter `signature(x = "AbstractTableDescriber", index = "numeric")`: returns the quote characters(s) for the data object being described

documented.d1Identifiers `signature(x = "AbstractTableDescriber")`: return a vector of dataONE identifiers associated with each table

documented.entityNames `signature(x = "AbstractTableDescriber")`: return a vector of entity names associated with each table

documented.sizes `signature(x = "AbstractTableDescriber")`: return a vector of data table sizes (in bytes)

Author(s)

Matthew Jones and Rob Nahf

Examples

```
showClass("AbstractTableDescriber")
```

CertificateManager-class

Class "CertificateManager"

Description

Certificate Manager organizes a set of methods to understand and, if necessary, manipulate the state of your DataONE identity, as represented by your CILogon client certificate.

Understanding how your identity is managed is important for working with DataONE, especially to avoid unexpected results. For example, depending your authorization status, searches may or may return only public records, or the full set of public and private records. Object and package retrievals might fail if some or all of the objects being retrieved are private. Creates and identifier reservations might fail because your authorization certificate is missing or expired.

Details

DataONE identifies you using CILogon-provided x509 certificates. DataONE has partnered with CILogon to provide a widely-accessible certificate issuing mechanism that allows DataONE users to use existing trusted institutional and public accounts.

CILogon recognizes many identity providers, including many universities and also Google, so most times users new to DataONE can get certificates using one of their existing accounts. For more information about the CILogon service, see http://www.cilogon.org/faq

X509 Certificates differ from typical username-password login schemes in that certificates can be used by more than one application, which is very useful when using more than one DataONE-enabled application. The certificates CILogon issues for DataONE are so-called "short-lived" certificates that currently expire 18 hours from the time of issuing. Typically you will want to download a fresh certificate the first time you interact with DataONE each day.

Objects from the Class

Objects can be created by calls of the form `new("CertificateManager", ...)` or `CertificateManager()`

Slots

`jClientIdManager`: Object of class `"jclassName"` ~~

Methods

downloadCert `signature(x = "CertificateManager")`: open the CILogin Certificate download page in the default browser

getCertExpires `signature(x = "CertificateManager")`: Show the date and time when the client certificate expires

isCertExpired `signature(x = "CertificateManager")`: Is the CILogon certificate expired?

obscureCert `signature(x = "CertificateManager")`: Provided mostly for troubleshooting, this method renames the CILogon certificate, so that future connections to DataONE are anonymous (as public). Note, when the client certificate is obscured, you will not be able to create objects to DataONE, or build D1Objects, which uses the certificate to fill out fields in the system metadata it creates.

restoreCert `signature(x = "CertificateManager")`: Restores a previously obscured certificate so that future interactions with the DataONE services are established using the certificate.

showClientSubject `signature(x = "CertificateManager")`: Returns the client subject (identity) according to DataONE, as as contained in the CILogon certificate. This is the same value that CILogon displays when you download a certificate. If the certificate is expired, the character string will be prefixed with "[EXPIRED]"

Author(s)

Matthew Jones and Rob Nahf

Examples

```
showClass("CertificateManager")
```

CertificateManager-methods

~~ *Methods for Function* `CertificateManager` *in Package* **dataone** ~~

Description

The CertificateManager provides methods for interacting with the client-side certificate. Building a CertificateManager object is done with this method.

Methods

`signature(... = "ANY")` this method returns a CertificateManager object.

Author(s)

Matthew Jones and Rob Nahf

D1Client-class *Class* "D1Client"

Description

A D1Client object represents the high-level interface to the DataONE system, and its methods mediate most all interaction with the system.

Objects from the Class

Objects can be created by calls of the form `new("D1Client", ...)` or using one of the constructor methods: `D1Client`

Slots

`endpoint`: Object of class `"character"` ~~

`mn.nodeid`: Object of class `"character"` ~~

`client`: Object of class `"jobjRef"` ~~

`session`: Object of class `"jobjRef"` ~~

Methods

convert.csv `signature(x = "D1Client")`: returns the dataframe serialized as a .csv

createD1Object `signature(x = "D1Client", d1Object = "D1Object")`: submits the objects of the Data Package for creation in the DataONE System.

createDataPackage `signature(x = "D1Client", dataPackage = "DataPackage")`: Creates the D1Objects contained in the DataPackage by calling the createD1Object() on each of the members, as well as assembling the resourceMap object from the recorded relationships, and calling create() on it as well. Any objects in the data map that have a dataUploaded value are assumed to be pre-existing in the system, and skipped.

d1IdentifierSearch `signature(x = "D1Client", solrQuery = "character")`: query the DataONE solr endpoint of the Coordinating Node, and return a character vector of identifiers. It expects a fully encoded character string as input (with lucene-reserved characters backslash escaped and url-reserved characters percent-encoded).

d1SolrQuery `signature(x = "D1Client", solrQuery = "character")`: query the DataONE solr endpoint of the Coordinating Node. It expects a fully encoded character string as input (with lucene-reserved characters backslash escaped and url-reserved characters %-encoded).

d1SolrQuery `signature(x = "D1Client", solrQuery = "list")`: query the DataONE solr endpoint of the Coordinating Node. the solrQuery list expects named parameters corresponding to SOLR url query constructs. Values of the list are expected to backslash escape any lucene reserved characters.

encodeSolr `signature(x = "D1Client", segment = "character")`: treating all special characters and spaces as literals, backslash escape special characters, and surround with double-quotes if necessary

encodeUrlQuery `signature(x = "D1Client", querySegment = "character")`: Encodes the characters of the input so they are not interpreted as reserved characters in url strings. Will also encode non-ASCII unicode characters.

encodeUrlQuery `signature(x = "D1Client", pathSegment = "character")`: Encodes the characters of the input so they are not interpreted as reserved characters in url strings. Will also encode non-ASCII unicode characters.

getCN `signature(x = "D1Client")`: ...

getD1Object `signature(x = "D1Client")`: retrieve an object as a D1Object from the DataONE system by its identifier

getEndpoint `signature(x = "D1Client")`: Get the URI endpoint of the CN service which D1Client is using. This value is determined from the env parameter

getMN `signature(x = "D1Client", nodeid = "ANY")`: ...

getMN `signature(x = "D1Client", nodeid = "character")`: ...

getMNodeId `signature(x = "D1Client")`: returns the identifier for the default Member Node

getPackage `signature(x = "D1Client", identifier = "character")`: retrieve a DataPackage from the DataONE system by its identifier, including all of it's members.

listMemberNodes `signature(x = "D1Client")`: list the nodes registered to the DataONE environment

reserveIdentifier `signature(x = "D1Client", id = "character")`: reserve an identifier on DataONE in preparation for building and submitting a new object.

setMNodeId `signature(x = "D1Client", id = "character")`: sets the default Member Node, to which future submissions will be directed. Note, D1Objects already built will be directed at the previous Member Node when createD1Object is called, since their systemMetadata specifies the old value.

Note

users should not provide the leading '?' to any query methods

The DataPackage describes the collection of data object and their associated metadata object, with the relationships and members serialized into a document stored under, and retrievable with, the packageId as it's own distinct object.

Members are created serially, and most errors in creating one object will interrupt the create process for the whole, resulting in some members will getting created, and the remainder not.

Author(s)

Matthew Jones and Rob Nahf

References

See d1_libclient_java documentation D1Client.create() "http://dev-testing.dataone.org:8080/hudson/job/d1_libclient_java/ws/d1_libclient_java/target/site/apidocs/org/dataone/client/D1Client.html#create"

Examples

```
showClass("D1Client")
## Not run:
    encodeSolr(client, "this & that")
    fullyEncodedPath <- paste0("cn/v1/object/",encodeUrlPath("doi:10.6085/AA/YBHX00_XXXITBDXMMR01_20040720
    fullyEncodedQuery <- paste0("q=id:",encodeUrlQuery(encodeSolr("doi:10.6085/AA/YBHX00_XXXITBDXMMR01_200

    d1IdentifierSearch(client,"q=%2Bspecies%20population%20diversity")
```

```
    d1SolrQuery(client,list(q="+species population diversity", fl="identifier"))
    d1SolrQuery(client,"q=%2Bspecies%20population%20diversity%26fl=identifier")

## End(Not run)
```

D1Client-methods ~~ *Methods for Function* D1Client *in Package* **dataone** ~~

Description

A D1Client object can be created in one of 3 ways, as described below. The user has a choice of environments and member nodes within the chosen environment when building the D1Client object.

Valid choices for env are: 'PROD' (the default), 'SANDBOX','STAGING', 'STAGING2' and 'DEV'

Methods

`signature(env = "ANY", mnNodeid = "ANY")` Creates a D1Client pointing to the production DataONE environment. It does not set a member node.

`signature(env = "character", mnNodeid = "ANY")` Creates a D1Client pointing to the environment specified, also without setting a default member node.

`signature(env = "character", mnNodeid = "character")` Creates a D1Client pointing to the environment specified, and setting the default member node for all content submission.

D1Object-class *Class* "D1Object"

Description

This class encapsulates a DataONE object and it's associated systemMetadata, and provides methods for building one for submission and methods for getting at its data and system metadata.

Objects from the Class

Objects can be created by calls of the form `new("D1Object", id, data, format, mnNodeId)`.

Slots

jD1o: Object of class "jobjRef" ~~

Methods

addData `signature(x = "DataPackage", d1object = "D1Object")`: Add the D1Object to the DataPackage's object store, usually in preparation for upcoming submission.

asDataFrame `signature(x = "D1Object", reference = "AbstractTableDescriber")`: return the D1Object's content as a dataFrame, using parsing instructions from the specific AbstractTableDescriber. see addition method implementation in `DataPackage-class`

asDataFrame `signature(x = "D1Object", reference = "ANY")`: return the D1Object's content as a dataFrame

asDataFrame `signature(x = "D1Object", reference = "D1Object")`: return the D1Object's content as a dataFrame, using parsing instructions contained in the reference metadata D1Object.

canRead `signature(x = "D1Object", subject = "character")`: returns TRUE if the subject provided has read permission on the D1Object, based on the local copy of the D1Object's AccessPolicy

createD1Object `signature(x = "D1Client", d1Object = "D1Object")`: Creates a D1Object on the MemberNode determined by the object's systemMetadata.

EMLParser `signature(d1Object = "D1Object")`: Return an EMLParser object for the given D1Object

getData `signature(x = "D1Object", id = "ANY")`: Returns the D1Object's content

getFormatId `signature(x = "D1Object")`: Get the format Identifier of the D1Object

getIdentifier `signature(x = "D1Object")`: Get the Identifier for the D1Object

setPublicAccess `signature(x = "D1Object")`: Grant read permission to the public on this D1Object, changing the system metadata locally. To be called before createD1Object(), otherwise it will not have any real affect.

Note

As of Feb 2013, there is only one AbstractTableDescriber subclass for parsing metadata: EMLParser. It can handle EML version 2.0.0 - 2.1.1 formatted metadata files. Other parsers should become available as separate packages that can be loaded as needed.

Author(s)

Matthew Jones and Rob Nahf

Examples

```
showClass("D1Object")


## Not run:
  ## asDataFrame
  df <- asDataFrame(data.package, dataMember.id)

  df <- asDataFrame(data.object, its.metadata)

  table.describer <- EMLParser(its.metadata)
  df <- asDataFrame(data.object, table.describer)

  df <- asDataFrame(data.object, sep="\t", ...)

  df <- asDataFrame(data.object)
```

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

DataPackage-class *Class* "DataPackage"

Description

An object representing a data package in DataONE, which consists of a series of data objects, the science metadata objects that documents them, and the resourceMap object that defines the relationships.

Objects from the Class

Objects can be created by calls of the form `new("DataPackage", packageId, jDataPackage)`.

Slots

`packageId`**:** Object of class `"character"` The identifier for the data package, that corresponds to the DataONE identifier for the resourceMap object. If a new package is being built, this should be a new identifier not already registered in DataONE.

`jDataPackage`**:** Object of class `"jobjRef"` A jJobRef to an instantiated java DataPackage object. This parameter is typically only used when retrieving an existing package from the DataONE system, as through the D1Client's 'getPackage' method.

Methods

addAndDownloadData `signature(x = "DataPackage", identifier = "character")`: downloads a pre-existing D1Object to the DataPackage, using the provided identifier string to retrieve from the DataONE system, and adds it to the local representation.

addData `signature(x = "DataPackage", d1object = "D1Object")`: Add the D1Object to the DataPackage's object store, usually in preparation for upcoming submission.

asDataFrame `signature(x = "DataPackage", reference = "character")`: return the content of the referenced DataPackage member as a dataFrame, using parsing instructions contained in the object's metadata. The metadata found via the DataPackage resource map.

contains `signature(x = "DataPackage", identifier = "character")`: Returns true if the specified object is a member of the package

createDataPackage `signature(x = "D1Client", dataPackage = "DataPackage")`: ...

getData `signature(x = "DataPackage", id = "character")`: Returns the content of the DataPackage member identified by id. see additional implementation in `link{D1Object-class}`

getIdentifiers `signature(x = "DataPackage")`: Return the identifiers of the package members, as defined by the ResourceMap

getMember `signature(x = "DataPackage", identifier = "character")`: Given the identifier of a member of the data package, return the D1Object representation of the member.

getSize `signature(x = "DataPackage")`: Get the count of D1Objects in the DataPackage

insertRelationship `signature(x = "DataPackage", metadataID = "character", dataIDs = "character")`: associate data Objects to the science metadata objects that describe them. Note that since the resource map that defines a package is separate from the items it associates, it is possible to use identifiers that have not been defined as members of the package, though not recommended.

removeMember `signature(x = "DataPackage", identifier = "character")`: removes a D1Object from the object map.

Note

A DataPackage object is a thin wrapper around the Java org.dataone.client.DataPackage class, exposing most methods, and adapting their parameters to remove the need to provide java-specific instances.

At least one method, "asDataFrame," is completely implemented within this package and has no corresponding java method.

Author(s)

Matthew Jones and Rob Nahf

Examples

```
showClass("DataPackage")

## Not run:
    ## example of instantiating a new DataPackage
    data_package <- new(Class="DataPackage",packageId="somePackageId")

    ## example of instantiating an existing DataPackage, through the
    ## fictional intermediate function foo()
    jD1Package <- foo()
    data_package <- new(Class="DataPackage", jDataPackage=jD1Package)

## End(Not run)
```

EMLParser-class *Class* "EMLParser"

Description

Class that implements methods to provide parsing instructions for asDataFrame for EML metadata documents v2.0.0 - v.2.1.1

Objects from the Class

Objects can be created by calls of the form `new("EMLParser", ...)`.

Slots

d1Object: Object of class "D1Object" ~~

xmlDocRoot: Object of class "XMLNode" ~~

Extends

Class "AbstractTableDescriber", directly.

Methods

data.characterEncoding `signature(x = "EMLParser", index = "numeric")`: ...

data.formatFamily `signature(x = "EMLParser", index = "numeric")`: ...

data.tableAttributeNames `signature(x = "EMLParser", index = "numeric")`: ...

data.tableAttributeOrientation `signature(x = "EMLParser", index = "numeric")`: ...

data.tableAttributeStorageTypes `signature(x = "EMLParser", index = "numeric")`: ...

data.tableAttributeTypes `signature(x = "EMLParser", index = "numeric")`: ...

data.tableFieldDelimiter `signature(x = "EMLParser", index = "numeric")`: ...

data.tableMissingValueCodes `signature(x = "EMLParser", index = "numeric")`: ...

data.tableQuoteCharacter `signature(x = "EMLParser", index = "numeric")`: ...

documented.d1Identifiers `signature(x = "EMLParser")`: ...

documented.entityNames `signature(x = "EMLParser")`: ...

documented.sizes `signature(x = "EMLParser")`: ...

Author(s)

Matthew Jones and Rob Nahf

See Also

`AbstractTableDescriber` for method descriptions

Examples

```
showClass("EMLParser")
```

---

tableDescriber.registry

Registry of D1 formatIds where TableDescriber subclasses register the metadata formats they can handle.

Description

Classes that inherit from TableDescriber should register themselves to this data structure when loaded, so it may be used by the `asDataFrame` methods.

Format

The format is: List of 4 $ eml://ecoinformatics.org/eml-2.1.1: chr "EMLParser" $ eml://ecoinformatics.org/eml-2.1.0: chr "EMLParser" $ eml://ecoinformatics.org/eml-2.0.1: chr "EMLParser" $ eml://ecoinformatics.org/eml-2.0.0: chr "EMLParser"

Author(s)

Matthew Jones and Rob Nahf

Examples

```
data(tableDescriber.registry)
```

Index