

Preservation Interoperability Standards

Reagan W. Moore
DICE Center
University of North Carolina
202 Manning Hall
Chapel Hill, NC 27599-3360
+1 919 962 9548
rwmooore@renci.org

Arcot Rajasekar
DICE Center
University of North Carolina
202 Manning Hall
Chapel Hill, NC 27599-3360
+1 919 962 9548
sekar@diceresearch.org

Mike Wan
DICE Group
University of California, San Diego
9500 Gilman Drive
San Diego, CA 92093
+1 858 534 5029
mwan@diceresearch.org

ABSTRACT

Preservation is based on properties related to authenticity, integrity, and chain of custody. These properties are maintained by policies and procedures that enable future users to understand and use the preserved collection. Preservation interoperability corresponds to the application of the preservation policies and procedures across independent preservation environments. This interpretation drives a set of interoperability standards that are based on the characterization of preservation policies, composition of preservation procedures, and validation of properties of the preservation environment. Policy-based data management systems provide the ability to define and enforce preservation properties. An example is the iRODS integrated Rule-Oriented Data System, <http://irods.diceresearch.org>.

Categories and Subject Descriptors

H.3.4 [Systems and Software] Distributed Systems — *policy-based data management*.

General Terms

Management, Design.

Keywords

Distributed rule engine, data grid, policy evolution, preservation.

1. INTRODUCTION

The properties that a preservation environment should possess are the result of a social consensus formed by the creators of the archive [1]. Properties may be related to preservation of authenticity, or trustworthiness, or original arrangement, or integrity, or completeness, or authoritativeness, or chain of custody. The properties are maintained over time by coding and enforcing policies that govern ingestion of records into the collection, management of the records, and collection access [2]. Each policy controls the execution of procedures that are needed to enforce the desired properties. A procedure may extract required metadata, or replicate a file, or implement a disposition mechanism. After the execution of each procedure, system state information is generated that tracks the results of applying the procedure, such as the metadata values that were extracted, or the location of the replicas, or the list of files for which the disposition policy was enforced. From this perspective, a preservation environment for digital data collections is characterized by the policies that translate to machine-executable procedures, which in turn require maintenance of state information that are self-consistently tracked over the lifetime of the collection.

Interoperability standards can be based on the set of policies and procedures that are needed to ensure preservation properties. Consider the following scenario:

- A record is extracted from an original preservation environment and migrated into a new preservation environment.
- The policies that govern the management of the records are migrated to the new preservation environment
- The procedures that manipulate the records are migrated to the new preservation environment
- The state information that tracks the results of applying the procedures are migrated to the new preservation environment.

An assertion can then be made that the properties of the archive can be maintained across both the original preservation environment and the new preservation environment. This assertion relies on the characterization of the preservation environment as a set of policies, procedures, and state information that can be quantified and moved independently of the choice of implementation technology.

In policy based data management systems, policies are implemented as machine-actionable rules [3]. The rules are composed by linking standard operations into workflows that are applied at remote storage systems. The state information generated by applying the workflows are stored in a metadata catalog that can be queried to verify that the desired properties have been maintained. In particular, a preservation environment is driven by the set of policies and procedures that define properties related to authenticity, original arrangement, integrity, chain of custody, and trustworthiness. Authenticity corresponds to tracking the provenance of the record and asserting the record is the same as the original submission. The original arrangement denotes the order in which records are submitted within a record series. Integrity is managed by testing whether the bits within the record have been corrupted and by using replicas to replace corrupted data. Chain of custody is managed by tracking where the files have been stored. Trustworthiness is validated through assessment criteria developed by standards organizations such as ISO MOIMS-rac [4].

In this note, we use the integrated Rule Oriented Data System (iRODS) [5] to explore interoperability standards that are needed for migration of records across preservation environments. We examine characterizations of preservation services, the policies that control application of the services, the fundamental

operations used to compose the preservation services, and the state information that is generated.

2. Preservation Services

A set of preservation services have been characterized at a high level by the California Digital Library [6] and are listed in Table 1. They correspond to operations that are performed upon each record during ingestion, and operations that enable discovery of records at a future date. It is possible to consider standard interoperability mechanisms that focus on the ability to apply these services on any preservation environment.

Table 1. High Level Preservation Services

1	Appraisal
2	Accession
3	Logical naming
4	File-based storage
5	Arrangement
6	Description
7	Repository instance
8	Policy configuration
9	Directory typing
10	File differencing
11	Checksums
12	Format typing
13	Containerization
14	Identifier resolver
15	Search

Examples of this approach include the development of standards for aggregating files in containers such as Bagit, standard naming conventions for logical naming of files, and standards for validating integrity (checksum algorithms).

3. Preservation Policies

The services listed in Table 1 do not address the administrative tasks that are needed to maintain a preservation environment. The ISO MOIMS-rac assessment criteria for trustworthiness can be expressed as a set of tasks that need to be executed to maintain a viable archive. In effect, a preservation policy is needed for each task to control the frequency at which the task is performed, define the records over which the task is executed, and define the information that must be maintained to provide compliance over time. A standard set of preservation policies can then be constructed that each preservation environment should enforce.

The 52 criteria listed in Appendix 1 are opportunities for development of interoperability standards. Each of the criteria should be implementable as a policy and associated procedures within a preservation environment. By also migrating the appropriate policies, procedures, and state information as records

are moved between systems, it will be possible to verify compliance with the original preservation properties.

The task can be simplified by defining generic functions that can be combined to implement the criteria. Twelve of the trustworthiness assessment criteria listed in Appendix 1 reduce to use of a template to define required information content, or to support formatting of information into a desired structure. Eleven of the criteria are enforced through the fundamental framework of a policy-based data management system that supports infrastructure independence. Seven of the criteria are addressed through parsing of audit trails. Six of the criteria are addressed through management of replicas. Three of the criteria are addressed through validation of checksums. The remaining thirteen criteria are implemented as specific procedures that are applied to each record under the control of a rule, such as association of metadata with a record, or the application of a specific process.

If we create generic functions that parse information using a template, or generate a package using a template, we can reduce the effort needed to implement the assessment criteria. Similarly, a generic function can be implemented that parses audit trails for specific information, a generic function can create replicas, and a generic function can validate checksums. We also need a generic function that chooses which preservation policies to implement based on a template that defines policy parameters.

From this perspective, the assessment criteria can be reduced to the application of twenty generic functions within a policy-based preservation framework. This approach is similar to the concept of infrastructure independence that is implemented in data grids. Infrastructure independence is based on virtualization mechanisms for logical naming of files, for interoperability across storage protocols, for interoperability across authentication environments, and for decoupling clients from storage systems through insertion of middleware. The actions of clients are interpreted as a combination of generic operations that are executed on behalf of the client by the data grid. Specifically, policy-based data management systems extend infrastructure independence to include policy virtualization, the ability to enforce policies on a collection independently of the choice of storage technology and independently of the choice of client.

The iRODS data grid provides support for policy virtualization through the installation of a rule engine and rule base at each storage location. The policies for the preservation environment are cast as rules. Each rule is defined for a specific event, governed by a condition, and executed as a workflow composed from basic operations called micro-services. Error recovery is executed through a recovery workflow. All requests for data at the storage system are processed through the rule engine. This ensures that the preservation policies are enforced independently of the choice of access client.

The generic preservation functions for implementing assessment criteria are listed in Table 2. The generic functions are executed within the iRODS infrastructure independence framework that provides the persistent logical name space, access controls, descriptive and provenance metadata support, persistent mapping from the logical name space to the storage location, management of replicas, data transport, and policy execution environment.

Table 2. Generic Preservation Functions

1	Parse information from document based on a template
2	Create a document based on structure defined by a template
3	Migrate a document between structures defined by two templates
4	Parse required policies from a template specifying rule parameters
5	Generate audit trails for requested operation
6	Parse audit trails to identify specified event/operation
7	Create replicas
8	Synchronize replicas
9	Create and validate checksums
10	Generate event-based notification
11	Associate required metadata with a record
12	Manage a rule base of preservation policies
13	Apply a required rule to a record
14	Provide unique identifiers for users, records, rules, micro-services
15	Authenticate all users
16	Assign users to roles
17	Authorize all operations based on roles
18	Version policies, micro-services, and state information
19	Manage a staging area for data ingestion
20	Support federation of preservation environments

4. Management Policy Framework

Most systems apply policies at the time of ingestion, and create a workflow that supports the validation of content, creation of an Archival Information Package, and storage/replication. However there are multiple additional places within a data management framework where policies should be enforced. Basically, every operation within a preservation environment should be controlled by an explicit policy. Within the iRODS data grid, policies can be defined at the policy enforcement points listed in Appendix 2. In many cases, the enforcements points correspond to application of policies before an operation is initiated and after an operation is completed.

The question for interoperability standardization is whether these policy enforcement points are sufficient, or whether additional policy enforcement points are needed. This is equivalent to asking whether all of the generic operations that need to be executed by preservation procedures have been trapped with pre-process and post-process policy enforcement points.

Table 3 shows the application of policies as iRODS rules when a file is ingested into the data grid. A representative set of rules is shown. A full-fledged archival system will have more processing steps with additional archive-centric and collection-centric procedures. Here we show a minimal set (with many of the operations being null operations) which still give a flavor of the

Table 3. Policies Invoked on File Ingestion

ApplyRule: acChkHostAccessControl GotRule: acChkHostAccessControl
ApplyRule: acSetPublicUserPolicy GotRule: acSetPublicUserPolicy
ApplyRule: acAclPolicy GotRule: acAclPolicy
ApplyRule: acSetRescSchemeForCreate GotRule: acSetRescSchemeForCreate execMicroSrvc: msiSetDefaultResc(demoResc,null)
ApplyRule: acRescQuotaPolicy GotRule: acRescQuotaPolicy execMicroSrvc: msiSetRescQuotaPolicy(off)
ApplyRule: acSetVaultPathPolicy GotRule: acSetVaultPathPolicy execMicroSrvc: msiSetGraftPathScheme(no,1)
ApplyRule: acPreProcForModifyDataObjMeta GotRule: acPreProcForModifyDataObjMeta
ApplyRule: acPostProcForModifyDataObjMeta GotRule: acPostProcForModifyDataObjMeta
ApplyRule: acPostProcForCreate GotRule: acPostProcForCreate
ApplyRule: acPostProcForPut GotRule: acPostProcForPut GotRule: acPostProcForPut GotRule: acPostProcForPut execMicroSrvc: msiSysReplDataObj(tgReplResc,null)
ApplyRule: acSetMultiReplPerResc GotRule: acSetMultiReplPerResc
ApplyRule: acSetRescSchemeForCreate GotRule: acSetRescSchemeForCreate execMicroSrvc: msiSetDefaultResc(demoResc,0)
ApplyRule: acPreprocForDataObjOpen GotRule: acPreprocForDataObjOpen
ApplyRule: acSetVaultPathPolicy GotRule: acSetVaultPathPolicy execMicroSrvc: msiSetGraftPathScheme(no,1)

types of workflow that one can easily configure in a production archive system.

In this example, 10 policy points are applied when ingestion occurs. The first three policies are related to authentication and authorization. The second three policies are related to resource management (e.g. choice of resource based on attributes of the ingested object, or quota enforcement) and arrangement. The next two policies are concerned with system metadata extraction and/or

assignment. The ninth policy in this list is concerned with the creation of the ingested digital object in the iRODS system and concerns all aspects for integrity and provenance (e.g. ownership) maintenance. The tenth policy provides a hook for post-processing of the ingested digital object as required by collection curator and/or the owner of the data object. The Post-process policy can be quite complicated, possibly distributed, and executed after a delay through a scheduling and queuing system. In this example, there are three policies for post-processing but only one is applicable (the other two had guard conditions that were not met by the properties of the digital object; the guard conditions can be made of combination of type, arrangement, ownership, size, location, etc.). The one post-processing policy that was applied created a replica of the ingested digital object in another physical resource. This policy in turn invoked other policies as part of its execution. The post-processing steps are shown below the line within Table 3.

The example shows that even a simple process such as the ingestion of a file into an archive should invoke explicit policies that govern the desired preservation properties. Each operation performed upon a preservation environment should be controlled by an explicit policy that is migrated with the record. An archivist in the future should be able to verify the policies under which a record was archived.

5. Micro-Services

Within a data grid, micro-services are chained together to implement each of the procedures required by the preservation environment. The number of micro-services can be much larger than the number of assessment criteria. By composing selected standard micro-services into a workflow chain, it becomes possible to implement a required management policy without having to write additional code. The number of micro-services currently supported within iRODS is on the order of 205. A partial list is shown in Appendix 3.

Micro-services are essential for creating procedures that can be migrated to new operating systems, ensuring that a specific policy can be applied in the future. Within iRODS, micro-services invoke a standard I/O library based on Posix I/O. For each storage system, a driver is written that maps from the standard I/O calls to the specific required storage protocol. This means the micro-services are independent of the choice of operating system, and will run equally well on Linux, Windows, or Mac operating systems. Given a standard set of micro-services, it will then be possible to migrate policies with records and ensure that desired preservation properties are maintained no matter which preservation environment is managing a set of records.

The expression of preservation procedures can be quantified by identifying the micro-services from which they are composed. Each micro-service should execute a well-defined standard function, such as file replication, or metadata extraction based on a template, or checksum validation, or query on state information. The standard operations within a preservation environment can then be quantified as standard micro-services. The migration of policies to a new preservation environment is simplified if the same standard operations are available within the new system.

A preservation environment should be characterized by the policies that are enforced, the procedures that are executed, and the standard operations that are performed. In addition, each

procedure generates standard state information, such as the location where the replica was made. It is possible to characterize a preservation environment, and base interoperability between preservation systems on the ability to apply and enforce preservation policies.

6. Summary

Preservation interoperability standards need to include support for preservation policies, preservation procedures, and preservation state information. Preservation policies can include assessment criteria that are periodically executed to verify properties about the preservation environment, such as authenticity, integrity, chain of custody, and trustworthiness. An initial list of the preservation policies and procedures that have been defined for the NARA Transcontinental Persistent Archive Prototype [7] has been provided. The goal is to show that the listed capabilities are sufficient and necessary for the implementation of a preservation environment.

7. ACKNOWLEDGEMENTS

The research results in this paper were funded by the NARA supplement to NSF SCI 0438741, "Cyberinfrastructure; From Vision to Reality" - Transcontinental Persistent Archive Prototype (TPAP) (2005-2008) and by the NSF Office of Cyberinfrastructure OCI-0848296 grant, "NARA Transcontinental Persistent Archive Prototype", (2008-2012). The iRODS technology development has been funded by NSF ITR 0427196, "Constraint-based Knowledge Systems for Grids, Digital Libraries, and Persistent Archives" (2004-2007) and NSF SDCI 0721400, "SDCI Data Improvement: Data Grids for Community Driven Applications" (2007-2010).

8. REFERENCES

- [1] Moore, R., "Building Preservation Environments with Data Grid Technology", *American Archivist*, vol. 69, no. 1, pp. 139-158, July 2006.
- [2] Moore, R. W., NIST White paper on "Policy-based Road Map for Digital Preservation Interoperability Framework: Connecting Digital Preservation Repositories", U.S. Workshop on Roadmap for Digital Preservation, Gaithersburg, Maryland, March 29-31, 2010.
- [3] Rajasekar, A., M. Wan, R. Moore, W. Schroeder, "A Prototype Rule-based Distributed Data Management System", HPDC workshop on "Next Generation Distributed Data Management", May 2006, Paris, France.
- [4] ISO Mission Operations Information Management System repository audit and certification, <http://www.digitalrepositoryauditandcertification.org/bin/view/Main/WebHome>
- [5] Rajasekar A., R. Moore, C-Y. Hou, Christopher L., R. Marciano, A. de Torcy, M. Wan, W. Schroeder, S-Y. Chen, L. Gilbert, P. Tooby, and B. Zhu, "iRODS Primer: integrated Rule-Oriented Data Systems", ISBN: 9781608453337, Morgan-Claypool Publishers, San Rafael, CA., January 2010.
- [6] California Digital Library, <http://www.cdlib.org/services/uc3/curation/>
- [7] NARA Transcontinental Persistent Archive Prototype, <http://www.dlib.org/dlib/july07/07inbrief.html>

Appendix 1. Preservation Tasks for Trustworthiness

	Criteria
1	Address liability and challenges to ownership/rights.
2	Identify the content information and the information properties that the repository will preserve.
3	Maintain a record of the Content Information and the Information Properties that it will preserve.
4	Specify Submission Information Package format (SIP)
5	Verify the depositor of all materials.
6	Verify each SIP for completeness and correctness
7	Maintain the chain of custody during preservation.
8	Document the ingestion process and report to the producer
9	Document administration processes that are relevant to content acquisition.
10	Specify Archival Information Package format (AIP)
11	Label the types of AIPs.
12	Specify how AIPs are constructed from SIPs.
13	Document the final disposition of all SIPs
14	Generate persistent, unique identifiers for all AIPs.
15	Verify uniqueness of identifiers.
16	Manage mapping from unique identifier to physical storage location.
17	Provide authoritative representation information for all digital objects.
18	Identify the file type of all submitted Data Objects.
19	Document processes for acquiring preservation description information (PDI)
20	Execute the documented processes for acquiring PDI.
21	Ensure link between the PDI and relevant Content Information.
22	Verify completeness and correctness of each AIP.
23	Verify the integrity of the repository collections/content.
24	Record actions and administration processes that are relevant to AIP creation.
25	Specify storage of AIPs down to the bit level.
26	Preserve the Content Information of AIPs.
27	Actively monitor the integrity of AIPs.
28	Record actions and administration processes that are relevant to AIP storage.
29	Prove compliance of operations on AIPs to submission agreement.
30	Specify minimum descriptive information requirements to enable discovery.
31	Generate minimum descriptive metadata and associate with the AIP.
32	Maintain link between each AIP and its descriptive information.

33	Enforce access policies.
34	Log and review all access failures and anomalies.
35	Disseminate authentic copies of records
36	Maintain replicas of all records, both content and representation information
37	Detect bit corruption or loss.
38	Report all incidents of data corruption or loss and repair/replace lost data
39	Manage migration to new hardware and media
40	Document processes that enforce management policies
41	Document changes to policies and processes
42	Test and evaluate the effect of changes to the repository's critical processes.
43	Synchronize replicas
44	Delineate roles, responsibilities, and authorization for archivist initiated changes
45	Maintain an off-site backup of all preserved information
46	Maintain access to the requisite Representation Information.
47	Maintain and correct problem reports about errors in data or responses from users.
48	Provide a search interface
49	Perform virus checks
50	Implement a staging area
51	Migrate records to new formats
52	Create and certify Dissemination Information Packages

Appendix 2. Policy Enforcement Points

Policy
Set policy for host access control
Create a new collection with name "childColl" under the parent collection "parColl"
Create default collections (home, trash)
Create a new user
Create collections in Data Grid zone
Pre-process for file delete
Delete the child collection "childColl" under the parent collection "parColl"
Delete home collection
Delete user
Delete collections in a Data Grid zone
Apply the "action" to the list of files that meet the specified condition
Set policy for checking permissions on registering a file
Post-process for collection create
Apply processing to file on copy
Post-process on file create

Post-process on resource creation
Post-process on token creation
Post-process for user create
Post-process for file delete
Post-process on resource deletion
Post-process on token deletion
Post-process for user delete
Post-process for modification of ACLs on data or collection
Post-process for modification of AVU metadata for data/collection/resource/user
Post-process on modification of collection metadata
Post-process on modification of data metadata
Post-process on resource modification
Post-process on resource group modification
Post-process for user modify
Post-process for user group modify
Post-process for object move
Post-process for file read or file read.
Apply processing to file on put
Post-process for collection delete
Pre-process for collection create
Pre-process for resource creation
Pre-process on token creation
Pre-process for user create
Pre-process for file open or read
Pre-process on resource deletion
Pre-process on token deletion
Pre-process for user delete
Pre-process for modification of ACLs on data or collection
Pre-process for modification of AVU metadata for data/collection/resource/user
Pre-process on modification of collection metadata
Pre-process on modification of data metadata
Pre-process on resource modification
Pre-process on resource group modification
Pre-process for user modify
Pre-process for user Group modify
Pre-process for moving a file
Pre-process for collection delete
Purge files satisfying condition on expiration time
Rename the Data Grid zone from the name “oldZone” to the name “newZone”
Specify number of copies per resource
Set the default number of threads for data transfers

Set policy for allowed operations by public
Pre-process on file create, define selection scheme for default resource
Set policy for assigning physical path name
Set policy for using trash can
Optimize the Postgresql database after waiting “arg1” specified time. See delayExec Micro-service

Appendix 3. Micro-services

Add Rules, state information mapping and function mapping tables to the Rule engine.
Pre-pend another rule file to the core rule file
Change the core rule file
Clear Rules, state information mapping and function mapping tables that were added to the Rule engine.
Display the currently loaded state variable name mappings
Display the currently loaded function name mappings
Display the currently loaded Rules
Apply all applicable Rules when executing a given Rule
Assign a value to a parameter
Break out of While, for and forEach loops
Do not retry any other applicable Rules for this action
Delay execution of micro-Services or Rules
Fail immediately, but recovery and retries are possible
Iterate over a row of tables or a list in a For loop
Execute for loop with initial step and end condition
If-then-else conditional branch
Fail with no recovery.
Sleep
No action
Remote execution of micro-Services or Rules
Succeed immediately
While loop
Write a line (with end of line) to stdout buffer
Write a string to stdout buffer
Close an opened data object
Create a data object
Lseek to a location in a file
Open a data object
Read an opened data object
Write
Check whether user is owner
Check whether permission is granted
Checksum a data object

Copy a file
Get a file (retrieve data from the collection)
Move a data object from one resource to another
Put a file into the collection
Put with additional options
Rename a data object
Replicate a file
Rsync a data between iRODS and local file
Trim the number of replicas
Delete a file
Return type of object: data, collection, resource
Stat an object
Register a physical file into iRODS
Create a collection
Replicate all files in a collection
Delete a collection
Remotely execute a command
Return conversion rate for currencies from one country to another, using web service provided by http://www.webserviceX.NET
Return a stock quotation using web service provided by http://www.webserviceX.NET
Return host name and details given an IP address, using the web service provided by http://ws.fraudlabs.com/
Return position and type of an astronomical object given a name using the NASA/IPAC Extragalactic Database (NED) web service at http://voservices.net/ws_v2_0/NED.asmx
Return an image buffer given a position and cutout size using the SDSS Image Cut Out web service at http://skyserver.sdss.org
Periodically vacuum a Postgres database
Add a user to a group
Commit the database transaction
Create a collection by administrator
Create a new user
Delete a collection by administrator
Delete a user
Execute a given general query structure and return results
Given a condition string create a query, execute it and return the values
Continue an unfinished query
Both make and execute a query
Given a select list and a condition list, creates a pseudo-SQL query
Rename a collection
Rename the local data grid zone
Roll back the database transaction
Commit Changes to the database.

Perform a SQL operation on a remote database but without returning resulting output.
Rollback (don't commit) changes to the database.
Store results in an iRODS DataObject
Access a remote database returning results in standardout.
Create an Xmsg packet, given required information
Receive an Xmsg packet
Send an Xmsg packet
Create a new Message Stream
Connect to the XMessage Server designated by an iRODS Environment file variable
Disconnect from the Xmessage Server
Send email!
Send stdout as email
Ingest object metadata from a AVU structure
Given a key and a keyValPair structure, extract the corresponding value
Print keyvalue pairs to stdout buffer
Remove object metadata using a AVU structure
Convert an array of strings to a string separated by %- signs
Convert %-separated keyvalue pair strings into keyValPair structure
Write keyword value pairs to stdout or stderr, using a specified separator
Add Dublin Core metadata fields to an object or collection
Extract NARA-style metadata from AVU triplets
Extract AVU information using a template
Free a memory buffer
Return the difference between two system time stamps, given in Unix format (stored in a string)
Return the system time for the metadata catalog
Return the local system time of an iRODS Server
Given a TagName get the value from a file in taggedformat (pseudoXML)
Convert a human readable date to a system timestamp
Load AVU metadata from a file
Load template file contents into Tag structure
Write a buffer to stdout or stderr
Write an integer to stdout or stderr
Set the access control policy.
Checksum a data object.
Set the policy for determining certain data cannot be deleted.
Do Not Check file path permission when registering
Set the policy to No trash can.
Specify the Copy to avoid

If the data has multiple copies, specify the preferred Copy to use
Get data type based on file name extension
Set the default storage resource
Set the scheme for composing the physical path in the vault to GRAFT_PATH.
Set the number of copies per resource to unlimited
Set a list of resources that cannot be used by a normal user directly.
Specify the parameters for determining the number of threads to use for data transfer.
Set a list of operations that can be performed by the user "public".
Set the scheme for composing the physical path in the vault to RANDOM.
Set the scheme for selecting the best resource to use
Set the resource from default
Sort the replica randomly when choosing which Copy to use
Stage the data object to the specified resource before operation.
Replicate a data object.
Copy metadata triplets from one object to another object
Create new user from information in a data object
Delete user based on information in a data object
Export metadata AVU triplets for a collection and its contents in a “ ”separated format
Retrieve Audit Trail information for a given action ID
Retrieve Audit Trail information by keywords in the comment field
Retrieve Audit Trail information for an object ID
Retrieve Audit Trail information by time stamp period
Retrieve Audit Trail information for a user ID
Gets ACL (Access Control List) for a collection in ” separated format
Return the object count and total disk usage of a collection
Retrieve metadata AVU triplets for a collection in “ ” separated format
Get ACL (Access Control List) for a data object in “ ” separated format
Get the Archival information Package of a data object in XML format
Retrieves metadata AVU triplets for a data object and return them

as an XML file
Retrieve metadata AVU triplets for a data object in “ ” separated format
Get user ACL for all objects and collections
Get information about user
Guess the data type of an object based on its file extension
Load ACL from information in a data object
Parses an object for new metadata AVUs
Modify user information from information in a data object
Recursively copy a collection and its contents including metadata
Set data type for an object
Given an XML object and an XSLT object, return the XML object after applying the XSLT transformation
Read data from an HDF file
Read data attribute from an HDF file
Close an HDF file
Open an HDF file
Read attributes of a group in an HDF file
Add a property and value to a property list. If the property is already in the list, its value is changed. Otherwise the property is added.
Clear a property list
Clone a property list, returning a new property list
Return true (integer 1) if the keyword has a property value in the property list, and false (integer 0) otherwise. The property list is unmodified.
Parse a string into a new property list. The existing property list, if any, is deleted
Get the value of a property in a property list. The property list is left unmodified
Create a new empty property list
Remove a property from the list
Set the value of a property in a property list. If the property is already in the list, its value is changed. Otherwise the property is added.
Convert a property list into a string buffer. The property list is left unmodified.
Return the local system time