Policy Workbook -iRODS 4.2
by Sheau-Yen Chen, Mike Conway, Jon Crabtree, Cal Lee, Sunitha Misra, Reagan W. Moore, Arcot Rajasekar, Terrell Russell, Isaac Simmons, Lisa Stillwell, Helen Tibbo, Hao Xu

Copyright © 2015 by the iRODS Consortium. All rights reserved.
Printed in the United States of America.

Published by the iRODS Consortium, 100 Europa Drive, Suite 540, Chapel Hill, North Carolina, 27517 USA.

September 2015

Acknowledgements

This research was supported by:


NSF SDCI 0910431, SDCI Data Improvement: Data Grids for Community Driven Applications (2007–2010).


NSF OCI 1032732, SDCI Data Improvement: Improvement and Sustainability of iRODS Data Grid Software for Multi-Disciplinary Community Driven Applications (2010-2012).


The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the National Archives and Records Administration (NARA), the National Science Foundation (NSF), or the U.S. Government.
Abstract

Policy-based data management systems such as the integrated Rule Oriented Data System, automate the enforcement of management policies, automate administrative tasks, and automate the validation of assessment criteria. This book presents policy sets applied in six types of data management applications: 1) data sharing; 2) student digital library; 3) production data centers; 4) preservation; 5) protected data management; and 6) NSF Data Management Plans.
Table of Contents

1 Introduction ................................................................................................................................. 1
  1.1 Policy Library............................................................................................................................ 8

2 Data Sharing Policy Set ............................................................................................................... 22
  2.1 Manage user creation (Policy 1) .............................................................................................. 22
  2.2 Manage user deletion (Policy 2) .............................................................................................. 23
  2.3 Manage renaming of a data grid (Policy 3) .............................................................................. 23
  2.4 Set the maximum number of I/O streams (Policy 4) .............................................................. 24
  2.5 Bypass permission checks for registering a file (Policy 5) ...................................................... 24
  2.6 Set policy for defining physical path name for a file (Policy 6) ............................................. 25
  2.7 Set number of execution threads used to process rules (Policy 7) ....................................... 25
  2.8 Set policy for processing files in bulk (Policy 8) .................................................................... 25
  2.9 Manage indexing of the system state catalog (Policy 9) ....................................................... 26
  2.10 Set storage quota policy (Policy 10) ..................................................................................... 26
  2.11 Manage selection of storage resource (Policy 11) .............................................................. 26

3 Data Management Policy Set (SILS LifeTime Library) ......................................................... 28
  3.1 Turn on storage quota enforcement (Policy 10) ..................................................................... 28
      3.1.1 Check for missing quotas ............................................................................................... 28
      3.1.2 Calculate total storage usage ....................................................................................... 29
      3.1.3 Identify persons who exceeded their quota ................................................................. 29
      3.1.4 Periodically update quota check .................................................................................. 30
  3.2 Manage selection of storage resource (Policy 11) ............................................................... 31
  3.3 Manage selection of storage resource for replication (Policy 12) ......................................... 31
  3.4 Enforce replication of each new file (Policy 13) .................................................................... 31
  3.5 Manage access control policy (Policy 14) ............................................................................ 32

4 Data Administration Policy Set (RDA Practical Policy working group) ......................... 33
  4.1 Data access control policies (Policy 14) ................................................................................. 33
      4.1.1 Find the User_ID associated with a User_name: ......................................................... 33
      4.1.2 Find the File_ID associated with a file name: ............................................................. 34
      4.1.3 Set write access control for a user: .............................................................................. 34
      4.1.4 Set operations that are allowable for the user "public" ............................................... 35
      4.1.5 Check the access controls on a file: ............................................................................. 35
  4.2 Data format control policies (Policy 15) ................................................................................. 36
      4.2.1 Set format conversion flag ............................................................................................ 37
      4.2.2 Invoke format conversion ............................................................................................. 37
      4.2.3 Identify and archive specific file formats from a staging area .................................... 38
  4.3 Notification Policies (Policy 16) ............................................................................................ 39
      4.3.1 Notify on collection deletion ....................................................................................... 39
      4.3.2 Notification of events .................................................................................................. 40
  4.4 Use agreement policies (Policy 17) ........................................................................................ 41
      4.4.1 Set receipt of signed use agreement ............................................................................. 41
      4.4.2 Identify users without signed use agreement ................................................................. 41
  4.5 Integrity policy (Policy 18) ...................................................................................................... 42
      4.5.1 Verify access controls on files ..................................................................................... 42
      4.5.2 Check integrity and number of replicas of files in a collection .................................... 44
  4.6 Metadata extraction (Policy 19) .............................................................................................. 47
5 Odum Data Preservation Policy set

5.1 Automate access restrictions (Policy 14) .................................................................61
  5.1.1 Set inheritance of access controls on a collection .................................................61
  5.1.2 Check whether a specific person has access to a collection .................................62
  5.1.3 Identify all persons with access to files in a collection .........................................62
  5.1.4 Identify files that can be accessed by an account ................................................63
  5.1.5 Delete access to files for a specified account ......................................................64
  5.1.6 Copy files, access control lists, and AVUs to a federated data grid .....................65
5.2 Normalize data to non-proprietary formats (Policy 15) ................................................67
  5.2.1 Detection of format type .......................................................................................67
  5.2.2 Automate format type detection ..............................................................................68
  5.2.3 Identify file format extensions in a collection .......................................................68
5.3 Creation of PREMIS event data (Policy 16) ................................................................69
  5.3.1 Creating PREMIS event information ....................................................................69
  5.3.2 Sending messages over AMQP .............................................................................70
5.4 Automation of user submission agreements (Policy 17) ..............................................73
  5.4.1 Staging of files with a user submission agreement ..............................................73
5.5 Automatic Checksums (Policy 18) ............................................................................74
  5.5.1 Creating a BagIt file ..............................................................................................74
5.6 Automated capture of Provenance/contextual metadata (Policy 19) ..........................76
  5.6.1 Provenance for administrative policies ...............................................................77
5.7 Federation – periodically copy data (Policy 20) ........................................................83
5.8 De-identification of Data (Policy 25) .........................................................................84
  5.8.1 BitCurator based processing .................................................................................84
5.9 Unique Identifiers for Data Sets (Policy 26) .............................................................99
  5.9.1 Assigning a Handle to a File .................................................................................99
  5.9.2 Registering files in DataONE registry ..................................................................100
5.10 Authentication identity management (Policy 27) ......................................................100
  5.10.1 Verify access controls on each file ....................................................................101
5.11 Automated Data Reviews (Policy 28) .................................................................101
  5.11.1 Metadata Review ...............................................................................................101
5.12  Mapping metadata across systems (Policy 29) ................................................................. 102
  5.12.1  Validate HIVE vocabularies .......................................................................................... 103
5.13  Export Datasets in Multiple Formats (Policy 30) ................................................................ 104
  5.13.1  Polyglot Format Conversion ......................................................................................... 104
5.14  Check for viruses (Policy 31) ............................................................................................. 104
  5.14.1  Scan files and flag infected objects .............................................................................. 105
5.15  Rule set management (Policy 32) ....................................................................................... 106
  5.15.1  Deploy rule sets ............................................................................................................. 106
5.16  Parse event trail for all persons accessing a collection (Policy 33) ...................................... 107

6  Protected Data Policy Sets .................................................................................................... 121
6.1  Check for presence of PII on ingestion (Policy 34) ............................................................. 123
6.2  Check for viruses on ingestion (Policy 31) .......................................................................... 123
  6.2.1  Scan files and flag infected objects .............................................................................. 124
  6.2.2  Migrate files that pass the virus check .......................................................................... 124
6.3  Check passwords for required attributes (Policy 35) .......................................................... 124
6.4  Encrypt data on ingestion (Policy 36) .................................................................................. 125
6.5  Encrypt data transfers (Policy 37) ....................................................................................... 126
6.6  Federation - control data copies (Policy 38) ......................................................................... 126
6.7  Federation - manage remote data grid interactions (Policy 32) .......................................... 128
  6.7.1  Updating rule base across servers ................................................................................... 128
6.8  Federation – Copy Data from staging area (Policy 20) ......................................................... 130
6.9  Federation- manage data retrieval (Policy 39) ...................................................................... 131
6.10  Generate checksum on ingestion (Policy 40) .................................................................... 133
6.11  Generate report of corrections to data sets or access controls (Policy 41) ......................... 133
6.12  Generate report for cost (time) required to audit events (Policy 42) ................................. 136
6.13  Generate report of types of protected assets (Policy 43) .................................................... 136
6.14  Generate report of all security and corruption events (Policy 44) ..................................... 137
6.15  Generate report of the policies applied to collections (Policy 45) ..................................... 137
  6.15.1  Deploy rule sets ............................................................................................................. 137
  6.15.2  Update rule sets ......................................................................................................... 138
  6.15.3  Print rule sets .............................................................................................................. 139
6.16  List all storage systems being used (Policy 46) .................................................................. 139
6.17  List persons who can access a collection (Policy 47) ....................................................... 139
6.18  List staff by position and required training courses (Policy 48) ...................................... 140
  6.18.1  Set position and training ............................................................................................. 140
  6.18.2  List staff by position and training .............................................................................. 141
6.19  List versions of technology that are being used (Policy 49) .............................................. 140
6.20  Maintain document on independent assessment of software (Policy 50) ......................... 142
6.21  Maintain log of all software changes, OS upgrades (Policy 51) ....................................... 142
  6.21.1  Version log files ......................................................................................................... 143
6.22  Maintain log of disclosures (Policy 52) .............................................................................. 143
6.23  Maintain password history on user name (Policy 53) ...................................................... 144
6.24  Parse event trail for all accessed systems (Policy 54) ....................................................... 146
6.25  Parse event trail for all persons accessing collection (Policy 33) ..................................... 146
6.26  Parse event trail for all unsuccessful attempts to access data (Policy 55) ......................... 146
6.27  Parse event trail for changes to policies (Policy 56) ......................................................... 146
6.28  Parse event trail for inactivity (Policy 57) .......................................................................... 147
6.29  Parse event trail for updates to rule bases (Policy 58) ...................................................... 147
6.30  Parse event trail to correlate data accesses with client actions (Policy 59) ...................... 147
7 Data Management Plan Example Rules ................................................................. 165

6.31 Provide test environment to verify policies on new systems (Policy 60) .......... 148
6.32 Provide test system for evaluating a recovery procedure (Policy 61) .............. 149
6.33 Provide training courses for users (Policy 62) .................................................. 149
6.34 Replicate data sets on ingestion (Policy 13) ...................................................... 149
6.35 Replicate iCAT periodically (Policy 63) .......................................................... 149
6.36 Set access approval flag (Policy 64) ................................................................. 150
   6.36.1 Restrict access for “Protected” data ............................................................ 151
6.37 Set access controls (Policy 14) ........................................................................... 153
   6.37.1 Set access controls after proprietary period ............................................... 153
6.38 Set access restriction until approval flag is set (Policy 65) ............................ 154
6.39 Set approval flag per collection for enabling bulk download (Policy 66) ....... 154
6.40 Set asset protection classifier for data sets based on type of PII (Policy 67) ... 155
6.41 Set flag for whether tickets can be used on files in a collection (Policy 68) .... 155
   6.41.1 Remove public and anonymous access ....................................................... 156
6.42 Set lockout flag and period on user name - counting number of tries (Policy 69) ... 156
   6.42.1 Set lockout period on user name ................................................................. 156
6.43 Set password update flag on user name (Policy 70) ......................................... 158
6.44 Set retention period for data reviews (Policy 71) ............................................. 160
6.45 Set retention period on ingestion (Policy 21) .................................................... 160
6.46 Track systems by type (server, laptop, router,...) (Policy 72) ......................... 161
6.47 Verify approval flags within a collection (Policy 73) ........................................ 161
6.48 Verify files have not been corrupted (Policy 18) ............................................. 162
6.49 Verify presence of required replicas (Policy 74) ............................................ 162
6.50 Verify that no controlled data have public or anonymous access (Policy 75) ... 162
   6.50.1 Restrict access to “Protected” data ............................................................ 162
6.51 Verify that protected assets have been encrypted (Policy 76) ......................... 163
   6.51.1 Check that files with ACCESS_APPROVAL = 0 are encrypted ................. 163

7 Data Management Plan Example Rules ................................................................. 165

7.1 Staffing policies (Policy 48) ............................................................................... 170
7.2 Cost reporting (Policy 24) .................................................................................. 170
7.3 Collection creation planning (Policy 45) ........................................................... 171
7.4 Instrument control (Policy 77) .......................................................................... 173
7.5 Event log for collection formation (Policy 54) ................................................... 174
7.6 Collection reports (Policy 41) .......................................................................... 175
7.7 Product formation (Policy 17) .......................................................................... 176
7.8 Data category management (Policy 78) ........................................................... 177
7.9 Re-using existing data (Policy 79) ................................................................. 178
7.10 Quality control (Policy 80) ............................................................................... 178
7.11 Analysis procedures (Policy 81) ...................................................................... 179
7.12 Analysis collaborations (Policy 82) ................................................................. 180
7.13 Data dictionary (Policy 29) .............................................................................. 181
7.14 Naming control (Policy 83) .............................................................................. 181
7.15 Data format control (Policy 16) ...................................................................... 181
7.16 Unique identifiers (Policy 27) ......................................................................... 182
7.17 Metadata standard (Policy 29) ........................................................................ 183
7.18 Metadata export (Policy 84) ........................................................................... 183
7.19 Collection creation system (Policy 85) ............................................................ 184
7.20 Collection size (Policy 86) ............................................................................... 186
7.21 Publication of original data (Policy 87) ............................................................ 186
7.22 Publication of data products (Policy 88) .......................................................... 187
7.23 Re-use policies (Policy 89) .................................................................................. 188
7.24 Distribution policies (Policy 90) ......................................................................... 189
7.25 Privacy access restrictions (Policy 14) ................................................................. 191
7.26 IPR restrictions (Policy 91) .................................................................................. 192
7.27 Web access policies (Policy 92) ........................................................................... 194
7.28 Data sharing system (Policy 93) .......................................................................... 195
7.29 Code distribution system (Policy 94) ................................................................. 195
7.30 Retention period (Policy 21) .............................................................................. 196
7.31 Curation plans (Policy 95) .................................................................................. 196
7.32 Archive system (Policy 96) .................................................................................. 197
7.33 Replication policy (Policy 13) ............................................................................ 198
7.34 Backup policy (Policy 97) ................................................................................... 199
7.35 Integrity verification (Policy 18) ......................................................................... 200
7.36 Technology management policies (Policy 49) ...................................................... 201
7.37 Metadata catalog management (Policy 9) ............................................................ 201
7.38 Transformative migration (Policy 15) ................................................................. 201

8 Verifying Policy Sets: .......................................................................................... 203
8.1 Analysis of the integrated Rule Oriented Data System ............................................ 206
8.2 Policy-enforcement points .................................................................................. 207
8.3 Client invocation of policy-enforcement points .................................................... 207
8.4 Procedures executed at each policy enforcement point ................................. 208

9 Summary: ........................................................................................................... 213

10 Acknowledgements: ......................................................................................... 213

11 References: ....................................................................................................... 213

Appendix A: Policy-enforcement Points ................................................................. 215
Appendix B: Client Invocation of Policy Enforcement Points ................................... 217
Appendix C: Micro-services .................................................................................... 220
Appendix D: Persistent State Variables ................................................................. 231
Appendix E: Protected Data Requirements ............................................................. 237
Appendix F: Mauna Loa Sensor Data DMP ............................................................... 241
1 Introduction

The DataNet Federation Consortium (DFC) infrastructure enables communities to implement their preferred data management application. Partners within the DFC have implemented data sharing environments, data publication systems (digital libraries), data preservation systems (archives), data distribution systems, and data processing systems (processing pipelines). The DFC supports each type of data management application by specifying a set of policies that enforce the desired purpose for the collection.

A data sharing environment focuses on:
- Unified name spaces for users, files, collections, metadata
- Access controls
- Hierarchical arrangement
- Integrity

A digital library focuses on:
- Controlled name spaces for files, collections, metadata
- Descriptive metadata standards
- Standard data format
- PREMIS event data

An archive focuses on:
- Authenticity
- Integrity
- Chain of Custody
- Original arrangement

A data distribution system focuses on:
- Caching
- Replication
- Synchronization
- Access controls

A processing pipeline focuses on:
- Controlled name spaces for users, files, collections, metadata, and procedures
- Sharing of procedures, files
- Access controls
- Provenance of workflows

Each of these types of data management applications can build upon common data grid infrastructure by choosing an appropriate set of policies and procedures. The policies determine when and where the procedures are executed. Within the integrated Rule Oriented Data System (iRODS) data grid, policies can be automatically enforced at policy enforcement points, or policies can be executed interactively by a user or grid administrator, or policies can be scheduled for deferred and periodic execution. The policy enforcement points typically control management policies. Deferred and periodic execution are used for administrative tasks. Interactive execution may be used to validate assessment criteria.
This book lists policy sets that have been implemented in an iRODS data grid, generated in academic classes on digital library, and provided by user communities. Figure 1 lists the basic concepts underlying policy-based data management.

Given a specific data management purpose, a collection can be assembled that has desired properties such as integrity, authenticity, and access controls. The properties themselves may have associated requirements such as completeness (all files in the collection have each property), correctness (incorrect values for metadata have been identified and eliminated), consensus (the properties represent the combined desire of the group assembling the collection), and consistency (the same metadata and data format standards have been applied to all files in the collection).

![Policy-based Data Management Concept Graph](image)

Each desired property is enforced by a set of policies, that determine when and where associated procedures are executed. Thus an integrity property may require policies for generating checksums and replicating files. The associated procedures are workflows composed by chaining together basic tasks or functions (also called micro-services). The functions apply basic operations such as generate a checksum, or replicate a file, or set the data type. The results of applying the functions are saved as persistent state information or metadata attributes on the files, users, storage systems, policies, and micro-services.
Clients interact with the system by requesting actions that are trapped at policy enforcement points (PEP). At each PEP, a rule base is examined to determine which policy to apply, and the associated procedure is executed. To implement assessment criteria, policies can be executed periodically to verify collection properties.

We consider policy sets for the following purposes:

- Data sharing, implemented in the standard integrated Rule Oriented Data System (iRODS) release [1].
- Digital library management, implemented in the School of Information and Library Science LifeTime Library [2].
- Distributed data management, implemented in the Research Data Alliance Practical Policy working group [3].
- Data preservation, implemented in the DataNet Federation Consortium.

For each policy set, we define a set of iRODS rules that can be used to enforce management policies, automate administrative functions, and validate assessment criteria. The rules are written in the iRODS rule language [4-5]. Each rule that is run interactively has a rule name, a rule body enclosed in braces that is written in the iRODS rule language, INPUT variables, and OUTPUT variables. An example rule to say “hello world” is:

```plaintext
Mytestrule {
    # rule to write hello world
    writeLine("stdout", "$userNameClient says hello world");
}
INPUT null
OUTPUT ruleExecOut
```

Note that “ruleExecOut” on an OUTPUT line will copy the output information written to "stdout" to the user's screen. This enables retrieval of information generated through interactive execution of a rule. If the rule is executed at a policy enforcement point or executed periodically, the output should be written to a log file and saved within the data grid. The session variable, "$userNameClient", contains the name of the person who executed the command. The result printed to the screen by running this rule from account rwmoore with the irule command is:

```
rwmoore says hello world
```

The following examples include rules that can be run interactively by a user, rules that are run by a data grid administrator, rules that are enforced at Policy-Enforcement Points, and rules that run periodically under rule engine control.
Rules that are applied at Policy-Enforcement-Points have a standard rule name related to the specific action that is being controlled. The INPUT variables are typically replaced with session variables that track who is executing an external action. The INPUT variables may also be set through queries on the metadata catalog. Rules can query a metadata catalog to retrieve information about the collection, the users, the storage systems, and user-defined metadata. In many of the following examples, a query is made to the metadata catalog, a “foreach” loop is then used to process the rows returned from the query, parameters are extracted from the row structure using a “.” operator, and information is output to a log file using a writeLine micro-service. More information on the iRODS rule language can be found at [http://irods.org](http://irods.org), and in the “iRODS Primer” [4].

Policies from all six policy sets are included in this document. There is substantial overlap between policies from the Practical Policy working group, the DFC preservation policy set, and the Data Management Plan set. The policies unique to the DFC preservation policy set require interaction with external systems, which are listed in Table 1. While many of the policies are supported within the iRODS data grid, policies may require the use of external technologies, such as the InCommon authentication system, the HIVE Helping Interdisciplinary Vocabulary Engineering system, the Polyglot format translation service, the Bitcurator data analysis system, and the Handle file identifier system. The policy sets are identifie by the number in the leftmost column. When policies overlap across the six example areas, the policy number can be used to identify related policies. A total of 97 policy sets have been defined.

Table 1. Comparison of policy sets for data sharing, LifeTime Library, RDA data management, DFC preservation, Protected Data and Data Management Plans.

<table>
<thead>
<tr>
<th>Policies</th>
<th>iRODS default policies for data sharing</th>
<th>sils LifeTime Library policies</th>
<th>rda Practical Policy WG policies for administration</th>
<th>odum policy set for preservation</th>
<th>hipaa Protected Data</th>
<th>dmp Data Management Plans</th>
<th>Supporting Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  User creation</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>2  User deletion</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>3  Rename data grid</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>4  Set number of I/O streams</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>5  Server Permission checks</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>6  Physical path name</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>7  Execution threads</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>8  Bulk processing</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>9  Catalog indexing</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>10 Storage quota</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>11 Select storage</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td>12 Select replication resource</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------------------------------------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Replicate files</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Access controls</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Data format control policies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>iRODS, Polyglot</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Notification policies</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>iRODS, message bus</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Use agreement policies</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Verify files have not been corrupted</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>iRODS, SHA-128</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Contextual metadata extraction policies</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Federation - periodically copy data</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Data retention policies</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Data disposition policies</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Restricted searching policies</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Storage cost reports</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>De-identification of data</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>Bitcurator, iRODS</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Applying unique identifiers to data sets</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>Handle, iRODS</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>Authentication protocols for repository users</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>In-Common, iRODS</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Automated metadata review</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Mapping metadata across systems</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>HIVE, iRODS</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Ability to export datasets in multiple formats</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>Polyglot, iRODS</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Check for viruses on ingestion</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>ClamScan, iRODS</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Federation - manage remote data grid interactions</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Parse event trail for all persons accessing collection</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>iRODS, operation s</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Check for presence of PII on ingestion</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>Bitcurator, iRODS</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Check passwords for required attributes</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Encrypt data on ingestion</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Encrypt data transfers</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Federation - control data copies</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Federation- manage data retrieval</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Generate checksum on ingestion</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Generate report by collection of corrections to data sets or access controls</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>Generate report for cost (time) required to audit events</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Generate report of types of protected assets present within a</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>iRODS</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Generate report of all security and corruption events</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>Generate report of the policies that are applied to the collections</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>List all storage systems being used</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>List persons who can access a collection</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>List staff by position and required training courses</td>
<td>X</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>List versions of technology that are being used</td>
<td>X</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>Maintain document on independent assessment of software</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>Maintain log of all software changes, OS upgrades</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>Maintain log of disclosures</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>Maintain password history on user name</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>Parse event trail for all accessed systems</td>
<td>X</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>Parse event trail for all unsuccessful attempts to access data</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>Parse event trail for changes to policies</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>Parse event trail for inactivity</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>Parse event trail for updates to rule bases</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>Parse event trail to correlate data accesses with client actions</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>Provide test environment to verify policies on new systems</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>Provide test system for evaluating a recovery procedure</td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>Provide training courses for users</td>
<td>X</td>
<td>Operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>Replicate iCAT periodically</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>Set access approval flag</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>Set access restriction until approval flag is set</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>Set approval flag per collection for enabling bulk download</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>Set asset protection classifier for data sets based on type of PII</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>Set flag for whether tickets can be used on files in a collection</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>Set lockout flag and period on user name - counting number of tries</td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Set password update flag on user name</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>--------------------------------------</td>
<td>----</td>
<td>---</td>
<td>-------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>Set retention period for data reviews</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>Track systems by type (server, laptop, router, ...)</td>
<td></td>
<td>X</td>
<td>iRODS, operations</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>Verify approval flags within a collection</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>Verify presence of required replicas</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>Verify that no controlled data collections have public or anonymous access</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>Verify that protected assets have been encrypted</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>Instrument Type</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>Data category</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>Use of existing data</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>Quality control</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>Analysis</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>Data sharing during analysis</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>Naming attributes</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>Metadata export</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>Collection location</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>Size</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>Make original data public</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>Make data products public</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>Re-use</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>Re-distribution</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>IPR</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>Web access</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>Data sharing system</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>Code distribution system</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>Curation</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>Archive</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>Backup frequency</td>
<td></td>
<td>X</td>
<td>iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Typically, there is more than one way to provide the functions needed for a specific policy, and more than one way to implement a policy. In practice, policies are needed to initialize environmental variables, to enforce management decisions, and to validate assessment criteria. Thus each policy area may require the implementation of a set of policies for each user group or collection.
1.1 Policy Library

To simplify writing the policies, a library of standard policy functions has been developed, called dfc-functions.re. The operations that are supported are:

   Add AVU metadata to a file
   *Path  The iRODS path to a file;
   *Attname The attribute name to be added
   *Attvalue The attribute value to be added
   *Aunit  The attribute value to be added
   *Status The return status ("0" if successful)

   Add AVU metadata to a collection
   *Coll   The iRODS collection name
   *Attname The attribute name to be added
   *Attvalue The attribute value to be added
   *Attunit The attribute unit to be added
   *Status The return status ("0" if successful)

   Add usage and name to a list in sorted order
   *Name  A name to be added to a list which is sorted by usage
   *Usage The usage associated with the name
   *Listnam The return list of names that is sorted
   *Listuse The return list of usage values associated with the names
   *Min   Set to the minimum usage value currently in the list
   *Num   The size of the list (fixed input value)

4. checkCollInput (*Coll)
   This checks whether the input variable is a collection.
   *Coll The name of the collection to check. Fails if collection does not exist.

5. checkFileInput (*File)
   This checks whether the input variable is a file.
   *File The name of the file to check. Fails if file does not exist.

   This checks whether a collection exists.
   *Attname The name of a metadata attribute that should be present for the collection. Created if missing with value "0".
   *Coll  The name of the collection that is being checked.
   *Lfile The name of the output buffer for error messages
   *Val   The value of the metadata attribute, set to zero if the attribute was missing

7. checkPathInput (*Path)
   This checks whether a valid path name exists.
   *Path The iRODS path name to be verified (collection/file).

8. checkRescInput (*Res, *Zone)
   This checks whether the input variable is a storage resource in zone *Zone.
   *Res  The name of a storage resource to be checked.
   *Zone The name of the iRODS zone which has the resource.

9. checkUserInput (*User, *Zone)
   This checks whether the input variable is a user in zone *Zone.
   *User The USER_NAME of a user.
   *Zone The USER_ZONE of a user.

10. checkZoneInput (*Zone)
    This checks whether the designated zone is accessible through federation.
11. contains (*list, *elem)
   Returns true if list contains the element
   *list The list that is checked.
   *elem The element string which is tested for presence in the list.

12. createCollections (*coll, *cs)
   Create a sub-collection for each entry in list *cs under *coll
   *coll The full path to the parent collection.
   *cs A list of subdirectories that are added to the parent collection.

   Create a list of length *Num with default *Val
   *Lista The list that is being created.
   *Num The number of default values to put in the list.
   *Val The default value for each list item.

   This creates a log collection and a log file.
   *Coll The full path to a collection.
   *Sub The subdirectory that is created if necessary to hold the log file.
   *Name The name of the log file to which a time stamp is appended
   *Res The storage resource where the log file is stored.
   *Lpath Returns the full path to the log collection (*Coll/*Sub)
   *Lfile Returns the name of the log file
   *L_FD Returns the file descriptor for the log file.

   This creates *N replicas on a list of resources.
   *N The number of replicas to create of a file.
   *Numrepl The number of storage resources included in the list of resources.
   *Lfile The output buffer name for writing error messages.
   *Ulist A list that is set to “1” when a replica exists on a storage resource
   *Rlist The corresponding list of storage replicas.
   *Jround An index into the list of storage resources for the starting resource to use for replication.
   *Resource The resource used as the source for the replica.
   *Coll The collection name of the file being replicated.
   *File The name of the file that is replicated.
   *NumRepCreated A counter that is incremented as replicas are created.

   This deletes a metadata attribute and value from a file.
   *Path The irods full path to a file.
   *Attname The attribute name that will be deleted.
   *Attvalue The attribute value that will be deleted.
   *Aunit The attribute units that will be deleted.
   *Status The return status result (“0” if successful).

17. ext(*p)
   Extracts extension by parsing string for letters after a dot
   *p The string that is being parsed.

18. findZoneHostName (*Zone, *Host, *Port)
   This returns the Host name and Port for a federated zone.

19. getCollections (*filePaths)
   Returns list of collections by deleting the file name
   *filePaths Converts a list of paths into a list of collections.
20. getFiles (*localRoot, *localPaths)
   Returns list of files by stripping *localRoot from list *localPaths
   *LocalRoot The collection name that is stripped from the input paths.
   *localPaths Returns the list of files
    This counts the number of files and total size in a collection.
    *Coll The full path to a collection.
    *colldataID The number and size is calculated for all files in the collection with DATA_ID > *colldataID.
    *Size Returns the total size of files in the collection.
    *Num Returns the number of files in the collection.
    This creates a list of storage resources used by files in a collection.
    *Coll The full path to a collection that is analyzed.
    *Rlist Returns a list of resources on which files were stored.
    *Ulist Returns a usage list initialized to “0”.
    *Lfile The output buffer to which information is written.
    *Num Returns the number of resources that were found.
    Check if collection exists and create if necessary.
    *Lpath The full path name for an iRODS collection.
    *Lfile The output buffer to which information is written.
    *Status Returns “0” if the collection does not exist.
24. isData (*Coll, *File, *Status)
    This checks whether a file already exists.
    *Coll The full path name for an iRODS collection.
    *File The name of a file that is tested for presence in the collection.
    *Status Returns “0” if the file does not exist.
    This modifies an existing AVU attribute on a data file.
    *Path The full path to a file in iRODS.
    *Attname The attribute name that is being modified with a new value or unit.
    *Attvalue The new value that is being inserted.
    *Aunit The new unit that is being inserted.
    *Status Returns the status of the operation.
    This selects a resource to use from a list of storage resources.
    *Rlist A list of storage resources.
    *Ulist Corresponding list of usage with value “1” if the storage resource has a replica.
    *Num The number of storage resources in the list.
    *Resource Returns a resource that does not store a replica.
    Generates an access event message and sends it using AMQP
    *AccessType Input type of access event.
*UserName	 Input name of user who caused the event.
*DataId	 Input DATA_ID of a file that was manipulated.
*Time	 Input date when the event occurred.
*Description	 Input description of the event.
*eventOutcome	 Input event outcome.
*Host	 Input address of host where the event information is sent.
*queue	 Input queue where the message is sent.

Generate a JSON document describing a link between objects.
   *DataId	 Input DATA_ID of file that was manipulated.
   *AccessId	 Input event identifier value.
   *host	 Input address of host where the information is sent.
   *queue	 Input queue where the message is sent.

   Creates a JSON document describing a related event between objects.
   *relationshipType	 Input type of relationship.
   *relationshipSubType	 Input subtype for relationship.
   *DataIds	 List of DATA_IDs for files that are related.
   *AccessIds	 List of access IDs for the files.
   *host	 Input address of host for sending a message.
   *queue	 Input queue where message is sent.

This updates a metadata attribute on a collection.
   *Coll	 Path to a collection whose metadata is modified.
   *Attr	 Collection attribute name whose value is modified.
   *OldValue	 Original value for attribute.
   *NewValue	 New value for attribute.
   *Lfile	 Name of buffer where information is written.

31. uploadFiles (*localRoot, *localPaths, *coll)
Moves files in *localPaths to the collection *coll
   *localRoot	 The collection name that is stripped from the input paths.
   *localPaths	 List of file path names.
   *coll	 Name of collection where files are copied.

This verifies checksums on the replicas for a file.
   *Coll	 Collection whose files will be checked for integrity.
   *File	 The file in the collection checked for replicas.
   *Lfile	 Name of output buffer where information is written.
   *Num	 Number of storage resources in the storage resource list.
   *Rlist	 List of storage resources used by the collection.
   *Ulist0	 A list that has been initialized to "0".
   *Ulist	 Returns list of resources that were used to store a replica.
   *Numr	 Returns the number of replicas that exist on the storage resources.
   *NumBad	 Returns the number of files that have a bad checksum.

The rule examples assume that the library of policy functions has been loaded into the configuration file, /etc/irods/server_config.json by addition to the re_rulebase_set:

"re_rulebase_set": [
    { "filename": "core,dfc-functions"
  } ]
The dfc-functions.re library of policy functions is listed below. Updated versions of the policy functions are available for download at http://github.com/DICE-UNC/policy-workbook/dfc-functions.re.

```re
    *Status = "1";
    # using micro-service from iPlant Collaborative
    msiSetAVU("-d", *Path, *Attname, *Attvalue, *Aunit);
    *Status = "0";
}

    # add metadata to a collection
    *Status = "1";
    # using micro-service from iPlant Collaborative
    *Status = "0";
}

    # insert new usage in list keeping top 10
    *S = 0;
    for (*I=0; *I<*Num; *I=*I+1) {
        *Val = elem(*Listuse,*I);
        *Use = double(*Val);
        if (*Use < *Usage) {
            for (*J=*Num-1; *J>*I; *J=*J-1) {
                *Jm1 = *J-1;
                *Use1 = elem(*Listuse,*Jm1);
                *Nam1 = elem(*Listnam,*Jm1);
                *Listuse = setelem("Listuse","*J",*Use1);
                *Listnam = setelem("Listnam","*J",*Nam1);
            }
            *Listuse = setelem("Listuse","*I",str(*Usage));
            *Listnam = setelem("Listnam","*I",*Name);
            *S = 1;
        }
    }
    if (*S == 1) {break;}
    *Min = double(elem(*Listuse,*Num-1));
}

cHECKCollInput(*Coll) {
    # check whether *Coll is a collection
    # fail if not a collection
    *Q = select count(COLL_ID) where COLL_NAME = *Coll;
    foreach (*R in *Q) {*Result = *R.COLL_ID;}
    if(*Result == "0") {
        writeLine("stdout","Input path *Coll is not a collection");
        fail;
    }
}
```

"
checkFileInput (*File) {
    # check whether *File is a file
    # fail if not a file
    *Q = select count(DATA_ID) where DATA_NAME = '*File';
    foreach (*R in *Q) {*Result = *R.DATA_ID;
        if(*Result == "0") {
            writeLine("stdout","Input *File is not a file");
            fail;
        }
    }
}

    # create metadata attribute on collection *Coll if it does not exist and initialize to 0
    # log creation event to a logfile *Lfile
    # return either 0 or prior value
    *Val = "0";
    *Query1 = SELECT COUNT(META_COLL_ATTR_NAME) where COLL_NAME = '*Coll' and META_COLL_ATTR_NAME = '*Attname';
    foreach (*Row1 in *Query1) {
        *Val = *Row1.META_COLL_ATTR_NAME;
    }
    if(int(*Val) == 0) {
        addAVUMetadataToColl(*Coll, *Attname, "0", ",", *Status);
        writeLine(*Lfile,"added TEST_DATA_ID attribute to collection *Coll");
    }
    *Query2 = select META_COLL_ATTR_VALUE where COLL_NAME = '*Coll' and META_COLL_ATTR_NAME = '*Attname';
    foreach(*Row2 in *Query2) {
        *Value = *Row2.META_COLL_ATTR_VALUE;
    }
}

checkPathInput (*Path) {
    # check whether *Path is a valid path
    # fail if not a valid path
    *Q = select count(DATA_ID) where DATA_NAME = '*File' and COLL_NAME = '*Coll';
    foreach (*R in *Q) {*Result = *R.DATA_ID;
        if(*Result == "0") {
            writeLine("stdout","Input *Path is not a valid path");
            fail;
        }
    }
}

checkRescInput (*Res, *Zone) {
    # local zone is defined by your irods_environment file
    if (*Zone != $rodsZoneClient) {
        # execute query in the remote zone
        findZoneHostName(*Zone, *Host, *Port);
        remote (*Host,"<ZONE>*Zone</ZONE>") {
            *Q1 = select count(RESC_ID) where RESC_NAME = '*Res';
            foreach (*R1 in *Q1) {*n = *R1.RESC_ID;
                if (*n == "0") {
                    writeLine("stdout","Remote resource *Res is not defined in zone *Zone");
                }
            }
        }
    }
writeLine("stdout", "Resource *Res exists in remote zone *Zone");
}
else {
  # query local zone
  *Q1 = select count(RESC_ID) where RESC_NAME = '*Res';
  foreach (*R1 in *Q1) {*n = *R1.RESC_ID;}
  if (*n == "0") {
    writeLine ("stdout", "Local resource *Res is not defined");
    fail;
  }
  writeLine ("stdout", "Resource *Res exists in local zone $rodsZoneClient");
}

checkUserInput (User, Zone) {
  # $rodsZoneClient is defined by your irods_environment file
  findZoneHostName (Zone, Host, Port);
  if ($rodsZoneClient != *Zone) {
    remote (Host, "<ZONE>*Zone</ZONE>") {
      *Q = select count(USER_ID) where USER_NAME = *User' and USER_ZONE = 'rodsZoneClient';
      foreach(*R in *Q) {*Result = *R.USER_ID;}
      if (*Result == "0") {
        writeLine ("stdout", "*User#$rodsZoneClient is not a user in zone *Zone");
        fail;
      }
      writeLine ("stdout", "*User is a user in zone *Zone");
    }
  } else {
    *Q1 = select count(USER_ID) where USER_NAME = *User';
    foreach (*R1 in *Q1) {*Result = *R1.USER_ID;}
    if (*Result == "0") {
      writeLine ("stdout", "*User not a user in zone $rodsZoneClient");
      fail;
    }
    writeLine ("stdout", "*User is a user in zone $rodsZoneClient");
  }
}

checkZoneInput (Zone) {
  *Q1 = select count(ZONE_ID) where ZONE_NAME = *Zone';
  foreach (*R1 in *Q1) {*n = *R1.ZONE_ID;}
  if (*n == "0") {
    writeLine ("stdout", "Remote zone *Zone is not federated");
    fail;
  }
}

contains(list, elem) {
  *ret = false;
  foreach (*e in list) {
    if (*e == *elem) {
      *ret = true;
    }
  }
  return *ret;
}
*ret = true;
}
}
*ret;
}

createCollections(*coll, *cs) {
foreach(*c in *cs) {
    msiCollCreate(""/"c","1","status");
}
}

createList(*Lista, *Num, *Val) {
# create a list with default values *Val
*Lista = list(*Val);
for (*I=1;*I<*Num;*I=*I+1) {
    *Lista = cons(*Val, *Lista);
}
}

# Create a log sub-directory within *Coll if it is missing
# Create a timestamped log file with the input file name *Name
msiGetSystemTime("human");
#============ create a collection for log files if it does not exist ============
*LPath = "Coll/Sub";
isColl (*LPath, "stdout", *Status);
if (*Status < 0) { fail; }
#============ create file into which results will be written ==============
*Lfile = "Path/Name-TimeH";
*Dfile = "destRescName=Res+++forceFlag=";
msiDataObjCreate(*Lfile, *Dfile, *L_FD);
}

*NumRepCreated) {
# create *N replicas for file *Coll/*File
# good replicas are in *Ulist
# good replica is in *Resource
# write actions to *Lfile
if(*N > 0) {
    writeLine("Lfile","File *Coll/*File is missing *N replicas");
    for(*I = 0; *I < *N; *I = *I+1) {
        #==pick resource to use for storing replica, round robin through storage systems without a
        replica ==
        *Check = false;
        *Loop = 0;
        for(*L = 0; *L < *Numrepl; *L = *L+1) {
            *Loop = *Loop + 1;
            if(*Loop >= 3) {
                break;
            }
        }
    }
}
*Stu = elem(*Ulist,*J);
if(*Stu == "0") {
*Resu = elem(*Rlist,*J);
msiDataObjRepl("*Coll/*File","destRescName=*Resu++++rescName=*Resource",*Status1);
*NumRepCreated = *NumRepCreated + 1;
*Ulist = setelem(*Ulist,*J,"1");
*Check = true;
*jround = *j + 1;
if(*jround >= *Numrepl) {
*jround = 0;
}
}
if(*Status1 < 0) {
*NumRepCreated = *NumRepCreated - 1;
writeLine("*Lfile","Unable to create a replica for *Coll/*File on resource *Resu");
*Check = false;
}
deleteAVUMetadata (*Path,*Attname,*Attvalue,*AUnit,*Status) {
*Str = "*Attname=*Attvalue";
msiString2KeyValPair(*Str,*Keyval)
msiRemoveKeyValuePairsFromObj (*Keyval,*Path,"‐d");
}
ext(*p) {
*b = trimr(*p,".");
*ext = if *b == *p then "no ext" else substr(*p,strlen(*b)+1,strlen(*p));
*ext;
}
findZoneHostName (*Zone,*Host,*Port) {
*Q1 = select ZONE_CONNECTION where ZONE_NAME = "*Zone";
foreach (*R1 in *Q1) {
*Conn = *R1.ZONE_CONNECTION;
msiSplitPathByKey (*Conn,":",*Host,*Port);
}
}
getCollections(*filePaths) {
*cs = list();
foreach(*p in *filePaths) {
*cs = list();
}
# Get the directory *p2 is not in the list *cs, add it as the 1st element
if(!contains(*cs, *p2) & & *p != *p2) {
    *cs = cons(*p2, *cs);
    #writeLine("stdout", ">>>>> cs = *cs 
\n");
}
}
}
*cs;
}

getFiles(*localRoot, *localPaths) {
# Construct a list *cs which finds file name by stripping *localRoot from list of *localPaths
*cs = list();
*localRootLen = strlen(*localRoot) + 1;
foreach(*p in *localPaths) {
    # use substr to chop off first *localRootLen from the absolute
    # path of the file - to get the next level of directory.
    *p1 = substr(*p, *localRootLen, strlen(*p));
    # Concatenate *p1 to the list *cs by adding it as its first element
    *cs = cons(*p1, *cs);
}
}
*cs;
}

# Only process files with DATA_ID > *colldataID
# Generate number and size
#======== *colldataID is the string identifier of the last file that has been checked ========
    *q1 = select count(DATA_NAME), sum(DATA_SIZE) where COLL_NAME like '*Coll%' and
    DATA_ID >= *colldataID';
#======== this counts all files that have not yet been checked including replicas
#============ use resources at which any files in the collection were stored ===========
    foreach(*r1 in *q1) {
        *num = *r1.DATA_NAME;
        *sizetotal = *r1.DATA_SIZE;
    }
    # end of retrieval of number and size
    *Size = double(*sizetotal);
    *Num = int(*num);
}

# generate a list of replicas *Rlist used by collection *Coll
# initialize a user list *Ulist to 0
# write resource names to log file *Lfile
#============ use resources at which any files in the collection were stored ===========
    *Query3 = select order_asc(DATA_RESC_NAME) where COLL_NAME like '*Coll%';
    *Num = 0;
    *Rlist = list();
    *Ulist = list();
    foreach(*R3 in *Query3) {
        *Str1 = *R3.DATA_RESC_NAME;
        *Rlist = cons(*Str1,*Rlist);
        *Ulist = cons("0",*Ulist);
        writeLine("*Lfile","Collection *Coll uses storage resource *Str1");
*Num = *Num + 1;
} # end of set up of list of resources

isColl (*LPath, *Lfile, *Status) {
    *Status = 0;
    *Query0 = select count(COLL_ID) where COLL_NAME = '*LPath';
    foreach (*Row0 in *Query0) {*Result = *Row0.COLL_ID;}
    if(*Result == "0") {
        msiCollCreate(*LPath, "1", *Status);
        if(*Status < 0) {
            writeLine("*Lfile","Could not create *LPath collection");
        } # end of check on status
    } # end of log collection creation
}

isData (*Coll, *File, *Status) {
    # Check whether a file already exists
    *Q = select count(DATA_ID) where COLL_NAME = '*Coll' and DATA_NAME = '*File';
    foreach (*R in *Q) {*Status = *R.DATA_ID;}
    *Status;
}

    # delete the original attribute value and add the new value to a file
    msiSplitPath (*Path, *Coll, *File);
    *Q1 = select META_DATA_ATTR_VALUE, META_DATA_ATTR_UNITS where DATA_NAME = '*File' and COLL_NAME = '*Coll' and META_DATA_ATTR_NAME = '*Attname';
    foreach (*R1 in *Q1) {*Avorig = *R1.META_DATA_ATTR_VALUE; }
    *Auorig = *R1.META_DATA_ATTR_UNITS; }
}

    # from list of resources *Rlist select a good copy *Ulist as source
    for(*J=0;*J<*Num;*J=*J+1) {
        if(elem(*Ulist,*J) == "1") {
            *Resource = elem(*Rlist,*J);
            break;
        } # end of selection of resource with valid copy
    } # end of loop over all resources
}

    # acsendAccess.r
    *AccessId = jsonEncode(genAccessId(*AccessType, *UserName, *DataId, *Time, *Description));
    *UserNameJson = jsonEncode(*UserName);
    *DescriptionJson = jsonEncode(*Description);
*DataIdJson = jsonEncode(*DataId);
*eventOutcome = jsonEncode(*eventOutcomeJson);
*msg='
{
  "messages" : [{
    "operation" : "create",
    "type" : "Event",
    "eventId" : {
      "eventIdType" : "URI",
      "eventIdValue" : "*AccessId"
    },
    "eventType" : "*AccessType",
    "eventDateTime" : "*Time",
    "linkingAgentIdentifier" : [
      {
        "linkingAgentIdentifierType" : "uri",
        "linkingAgentIdentifierValue" : "*UserNameJson"
      }
    ],
    "linkingObjectIdentifier" : [
      {
        "linkingObjectIdentifierType" : "uri",
        "linkingObjectIdentifierValue" : "*DataIdJson"
      }
    ],
    "eventDetail" : "*DescriptionJson",
    "eventOutcomeInformation" : "*eventOutcomeJson"
  }
}';
amqpSend(*host, *queue, *msg);
*AccessId;
}

sendLinkingEvent (*DataId, *AccessId, *host, *queue) {
  # aclinkEvent.r
  *DataIdJson = jsonEncode(*DataId);
  *msg='
  {
    "messages" : [{
      "operation" : "union",
      "objectIdentifierType" : "URI",
      "objectIdentifierValue" : "*DataIdJson",
      "linkingEventType" : "URI",
      "linkingEventIdentifierValue" : "*AccessId"
    }
  }
}';
amqpSend(*host, *queue, *msg);
}

  # acrelatedEvent.r
  *relationshipType = jsonEncode(*relationshipTypeJson);
  *relationshipSubType = jsonEncode(*relationshipSubTypeJson);
  *msg='


```plaintext
{
"messages": [
  
  "operation": "create",
  "type": "Relationship",
  "relationshipType": "relationshipTypeJson",
  "relatedEventIdentifier": [
    foreach(*DataId in *DataIds) {
      *DataIdJson = jsonEncode(*DataId);
      *msg = *msg ++ '{
        "relatedObjectIdentifierType": "URI",
        "relatedObjectIdentifierValue": *DataIdJson,
        "relatedObjectIdentifierSequence": "not applicable"
      },
    }
    if (substr(*msg, strlen(*msg) - 1, strlen(*msg)) == ",") {
      *msg = trimr(*msg, ",");
    }
    *msg = *msg ++ '],
  "relatedEventIdentifier": [
    foreach(*AccessId in *AccessIds) {
      *msg = *msg ++ '{
        "relatedEventIdentifierType": "URI",
        "relatedEventIdentifierValue": *AccessId,
        "relatedEventIdentifierSequence": "not applicable"
      },
    }
    if (substr(*msg, strlen(*msg) - 1, strlen(*msg)) == ",") {
      *msg = trimr(*msg, ",");
    }
    *msg = *msg ++ ']
  }
],
amqpSend(*host, *queue, *msg);
}

# For collection *Coll, update *Attr from *OldValue to *NewValue
# Log operations to *Lfile
*Str1 = "*Attr=*OldValue";
msiString2KeyValPair(*Str1, *kvp1);
msiRemoveKeyValuePairsFromObj(*kvp1, *Coll, "‐C");
*OldValue = *NewValue;
*Str2 = "*Attr=*NewValue";
msiString2KeyValPair(*Str2, *kvp);
msiAssociateKeyValuePairsToObj(*kvp, *Coll, "‐C");
writeLine("*Lfile", "Reset *Attr to *NewValue for collection *Coll");
}

uploadFiles(*localRoot, *localPaths, *coll) {
# Function uploadFiles takes in an array listing of files *localPaths, and copies the files
# to the given location *coll, in the grid. It creates a collection if it doesn't exist.
# uploadFiles: input string * input list string * input string -> integer
*fs = getFiles(*localRoot, *localPaths);
*cs = getCollections(*fs);
writeLine("stdout", "*coll, *localPaths, *fs, *cs");
createCollections(*coll, *cs);
}
for(*i=0;*i<size(*fs);*i=*i+1) {
    *obj = elem(*fs,*i);
    msiDataObjPut("*coll/*obj", "demoResc", "localPath=*lf++++forceFlag=", *status);
}

    # for file *Coll/*File and list of possible replica resources *Rlist
    # *Num is the number of resources
    # *NumBadFiles is a running total of the number of corrupted files
    # generate resource user list *Ulist of resources with good replicas
    *Query5 = select DATA_REPL_NUM,DATA_CHECKSUM,DATA_RESC_NAME where COLL_NAME = *Coll' and DATA_NAME = *File;
    *Numr = 0;
    *Ulist = *Ulist0;
    foreach(*Row5 in *Query5) {
        *Numr = *Numr + 1;
        *Repln = *Row5.DATA_REPL_NUM;
        *Chk = *Row5.DATA_CHECKSUM;
        *Rescn = *Row5.DATA_RESC_NAME;
        msiDataObjChksum("*Coll/*File", "replNum=*Repln++++forceChksum=", *Chkf);
        if(int(*Chk) == 0) {
            *Chk = *Chkf;
        } # end of set of checksum if not available
    } # end of loop over resources
    # end of processing good checksum
    if(int(*Chk) == int(*Chkf)) {
        for(*J=0;*J<*Num;*J=*J+1) {
            if(elem(*Rlist,*J) == *Rescn) {
                *Ulist = setelem(*Ulist,*J,"1");
                break;
            } # end of set of *Ulist for resource
        } # end of loop over resources
    } # end of processing good checksum
    # check whether checksum is correct, delete file if bad checksum ========
    if (int(*Chk) != int(*Chkf)) {
        writeLine("*Lfile","Bad checksum for replica *Repln of file *Coll/*File.");
        *NumBad = *NumBad + 1;
        *Ulist = setelem(*Ulist,*J,"0");
    } # end of processing a bad checksum
} # end of loop over replicas for a logical file
2 Data Sharing Policy Set

The iRODS Data grid distribution comes with 11 default policies that implement a data sharing environment. These policies are provided in a rule base, and are invoked automatically at policy-enforcement points within the data grid middleware. Actions initiated by clients are trapped at the policy-enforcement points, the rule base is accessed to determine the appropriate policy to apply, and an associated procedure is executed to enforce the policy.

The policies invoked at these enforcement points in the standard iRODS release are given a name that corresponds to the policy-enforcement point (typically starting with “ac”. In iRODS version 4.0.3 there are 70 standard policy enforcement points. Additional policy enforcement points can be plugged into the architecture to control new actions.

2.1 Manage user creation (Policy 1)

This policy is invoked when a new user is created. The rule creates a home directory and a trash directory for each new user account, and adds the account to the user group “public”. If the account is “anonymous”, the home directory and trash directories are not created. The rule uses session variables to identify the data grid zone name ($rodsZoneProxy) and the account name ($otherUserName). Note that there are two versions of the acCreateUserF1 rules. If the condition for the first rule is not satisfied, the second version of the rule is executed. If a task fails, the micro-service listed after the “:::” separator is executed. Thus interactions with the metadata catalog are “rolled back” if the registration attempt fails. The policy includes invocation of pre-processing and post-processing rules for user creation. This policy is distributed within the iRODS core.re file.

The rule is available at
https://github.com/irods/irods/blob/master/packaging/core.re.template

```
acCreateUser {
    acPreProcForCreateUser;
    acCreateUserF1;
    acPostProcForCreateUser; }
acCreateUserF1 {
    ON($otherUserName == "anonymous") {
        msiCreateUser ::: msiRollback;
        msiCommit; } }
acCreateUserF1 {
    msiCreateUser ::: msiRollback;
    acCreateDefaultCollections ::: msiRollback;
    msiAddUserToGroup("public") ::: msiRollback;
    msiCommit; }
acCreateDefaultCollections { acCreateUserZoneCollections; }
acCreateUserZoneCollections { 
    acCreateCollByAdmin("/++$rodsZoneProxy++/home", $otherUserName);
    acCreateCollByAdmin("/++$rodsZoneProxy++/trash/home", $otherUserName); }
acCreateCollByAdmin("parColl,"childColl) { msiCreateCollByAdmin(*parColl,*childColl); }
```
2.2 Manage user deletion (Policy 2)

This policy is invoked when a user account is deleted. The rule deletes the home and trash collections associated with a user account. The rule uses session variables to identify the data grid zone name ($rodsZoneProxy) and the account name ($otherUserName). Note that preprocessing policies (acPreProcForDeleteUser) and postprocessing policies (acPostProcForDeleteUser) can also be defined. These might be used to migrate files to an archive, or send e-mail to the user about the disposition of the files. This policy is distributed within the iRODS core.re file.

The rule is available at
https://github.com/irods/irods/blob/master/packaging/core.re.template

acDeleteUser{
    acPreProcForDeleteUser;
    acDeleteUserF1;
    acPostProcForDeleteUser;
}

acDeleteUserF1{
    acDeleteDefaultCollections :: msiRollback;
    msiDeleteUser :: msiRollback;
    msiCommit;
}

acDeleteDefaultCollections{
    acDeleteUserZoneCollections;
}

acDeleteUserZoneCollections{
    acDeleteCollByAdmin("/"++$rodsZoneProxy++"/home",$otherUserName);
    acDeleteCollByAdmin("/"++$rodsZoneProxy++"/trash/home",$otherUserName);
}

acDeleteCollByAdmin(*parColl,*childColl) {
    msiDeleteCollByAdmin(*parColl,*childColl);
}

acPreProcForDeleteUser {}
acPostProcForDeleteUser {}

2.3 Manage renaming of a data grid (Policy 3)

This policy is invoked when an administrative command is executed to rename a data grid. The rule renames all of the collections within the original data grid. The rule uses two input parameters to identify the original zone name (*oldZone) and the new zone name (*newZone). Both the name of the collection representing the zone and the zone name are reset. The string concatenation operator “++” is used to create the home data grid collection from the home data grid name. This policy is distributed within the iRODS core.re file.

The rule is available at
https://github.com/irods/irods/blob/master/packaging/core.re.template

acRenameLocalZone(*oldZone,*newZone) {
    msiRenameCollection("/"++str(*oldZone)++"","newZone") :: msiRollback;
    msiRenameLocalZone(*oldZone,"newZone") :: msiRollback;
    msiCommit;
}
2.4 Set the maximum number of I/O streams (Policy 4)
This policy is invoked when file transport is done from a storage resource. The policy controls the number of I/O streams that are used to move files across a network. The rule supports conditions based on the session variable $rescName so that different policies can be set for different resources. Only one function can be used for this rule:

```
msiSetNumThreads(sizePerThrInMb, maxNumThr, windowSize)
```

This sets the number of threads and the tcp window size. The number of threads is based on the input parameter sizePerThrInMb (size per thread in Mbytes). The number of threads is computed using:

```
numThreads = fileSizeInMb / sizePerThrInMb + 1
```

where sizePerThrInMb is an integer value in MBytes. It also accepts the word "default" which sets sizePerThrInMb to a default value of 32

```
maxNumThr - The maximum number of threads to use. It accepts integer values up to 16. It also accepts the word "default" which sets maxNumThr to a default value of 4. A value of 0 means no parallel I/O.
This can be helpful to get around firewall issues.
```

```
windowSize - the tcp window size in Bytes for the parallel transfer. A value of 0 or "default" means a default size of 1,048,576 Bytes.
```

The msiSetNumThreads function must be present or no parallel threads will be used for all transfers. This policy is distributed within the iRODS core.re file.

The rule is available at
https://github.com/irods/irods/blob/master/packaging/core.re.template

```
acSetNumThreads {msiSetNumThreads("default", "16", "default"); }
```

2.5 Bypass permission checks for registering a file (Policy 5)
This policy is invoked when files are registered into the data grid. The rule determines whether file path permissions are checked when registering a physical file path using commands such as ireg. The rule also sets the policy for checking the file path when unregistering a data object without deleting the physical file. Normally, a rodsuser account cannot unregister a data object if the physical file is located in a resource vault. The msiNoChkFilePathPerm allows this check to be bypassed. Only one function can be called:

```
msiNoChkFilePathPerm() - Do not check file path permission when registering a file. WARNING - This function can create a security problem if used.
```

This policy is distributed within the iRODS core.re file.

The rule is available at
https://github.com/irods/irods/blob/master/packaging/core.re.template

```
acNoChkFilePathPerm {msiNoChkFilePathPerm(); }
```
2.6 Set policy for defining physical path name for a file (Policy 6)
This policy is invoked before a file is stored in a file system. The rule defines the physical path that will be used within the iRODS resource vault. Two functions can be called:

msiSetGraftPathScheme(addUserName, trimDirCnt) - Set the VaultPath scheme to GRAFT_PATH - graft (add) the logical path to the vault path of the resource when generating the physical path for a data object. The first argument (addUserName) specifies whether the userName should be added to the physical path. e.g. $vaultPath/$userName/$logicalPath. "addUserName" can have two values - yes or no. The second argument (trimDirCnt) specifies the number of leading directory elements of the logical path to trim. A value of 0 or 1 is allowable. The default value is 1.

msiSetRandomScheme() - Set the VaultPath scheme to RANDOM meaning a randomly generated path is appended to the vaultPath when generating the physical path. e.g., $vaultPath/$userName/$randomPath. The advantage with the RANDOM scheme is renaming operations (imv, irm) are much faster because there is no need to rename the corresponding physical path. The default is the GRAFT_PATH scheme with addUserName == no and trimDirCnt == 1. Note: if trimDirCnt is greater than 1, the home or trash directory name will be taken out. This policy is distributed within the iRODS core.re file.

The rule is available at
https://github.com/irods/irods/blob/master/packaging/core.re.template

acSetVaultPathPolicy {msiSetGraftPathScheme("no","1");}

2.7 Set number of execution threads used to process rules (Policy 7)
This policy specifies the number of processes to use when running jobs in the irodsReServer. The irodsReServer can multi-task such that one or two long running jobs cannot block the execution of other jobs. One function can be called:

msiSetReServerNumProc(numProc) - numProc can be "default" or a number in the range 0-4. A value of 0 means no forking. The value of numProc will be set to 1 if "default" is input.

This policy is distributed within the iRODS core.re file.

The rule is available at
https://github.com/irods/irods/blob/master/packaging/core.re.template

acSetReServerNumProc {msiSetReServerNumProc("default");}

2.8 Set policy for processing files in bulk (Policy 8)
This rule sets the policy for executing the post processing put rule (acPostProcForPut) for bulk put operations. Since the bulk put option is intended to improve the upload speed, executing the acPostProcForPut for every file will slow down the the upload. This rule provides an option to turn the postprocessing off. Only one function can be called:
msiSetBulkPutPostProcPolicy (flag) - This micro-service sets whether the acPostProcForPut rule will be run on bulk put. Valid values for the flag are:
"on" - enable execution of acPostProcForPut.
"off" - disable execution of acPostProcForPut (default).
This policy is distributed within the iRODS core.re file.

The rule is available at https://github.com/irods/irods/blob/master/packaging/core.re.template

acBulkPutPostProcPolicy {msiSetBulkPutPostProcPolicy("off"); }  

2.9 Manage indexing of the system state catalog (Policy 9)
This rule controls the automated indexing of the metadata catalog. In the rule example, the indexing is delayed until a future time specified by the variable *arg1. Valid delay examples for *arg1 are:
"<PLUSET>1s</PLUSET>" – delay execution for one second
"<PLUSET>1m</PLUSET>" – delay execution for one minute
"<PLUSET>1h</PLUSET>" – delay execution for one hour
"<PLUSET>1d</PLUSET>" – delay execution for one day
"<PLUSET>1y</PLUSET>" – delay execution for one year
"<EA>ils.renci.org</EA>" – host address where execution is performed
This policy was provided in iRODS version 3.3, but has been deprecated in iRODS version 4.x.

acVacuum(*arg1) { delay(*arg1) { msiVacuum; } }

2.10 Set storage quota policy (Policy 10)
This rule can be used to turn on resource quota enforcement. The maximum storage space for each user can be set using the administrator command, iadmin. Quotas can be set for users and for groups of users, for either the total allowed storage or for the storage on a specific storage system. Only one function can be called:
msiSetRescQuotaPolicy() - This micro-service sets whether the Resource Quota should be enforced. Valid values for the flag are:
"on" - enable Resource Quota enforcement,
"off" - disable Resource Quota enforcement (default).
This policy is distributed within the iRODS core.re file.

The rule is available at https://github.com/irods/irods/blob/master/packaging/core.re.template

acRescQuotaPolicy {msiSetRescQuotaPolicy("on"); }  

2.11 Manage selection of storage resource (Policy 11)
This policy is invoked when creating a data object. The rule defines how resources are selected for storing files. This is a preprocessing rule that is executed before the object is created. It can be used to set the resource selection scheme when processing the put, copy and replicate operations. Currently, three preprocessing functions can be used by this rule:
• msiSetNoDirectRescInp(rescList) - sets a list of resources that cannot be used by a normal user directly. More than one resource can be input using the character "%" as separator. e.g., resc1%resc2%resc3. This function is optional, but if used, should be the first function to execute because it screens the resource input.

• msiSetDefaultResc(defaultRescList, optionStr) - sets the default resource. This function is no longer mandatory, but if it is used, if should be executed right after the screening function msiSetNoDirectRescInp.

  defaultResc - the resource to use if no resource is input. A "null" means there is no defaultResc. More than one resource can be input using the character "%" as separator.

  optionStr – Value can be "forced", "preferred" or "null". A "forced" input means the defaultResc will be used regardless of the user input. The forced action only applies to users with normal privilege, “rodsuser”.

• msiSetRescSortScheme(sortScheme) - set the scheme for selecting the best resource to use when creating a data object.

  sortScheme - The sorting scheme. Valid schemes are "default", "random", "byLoad" and "byRescClass". The "byRescClass" scheme will put the cache class of resource on the top of the list. The "byLoad" scheme will put the least loaded resource on the top of the list. In order to work properly, the Resource Monitoring system must be switched on in order to pick up the load information for each server in the resource group list. The scheme "random" and "byRescClass" can be applied in sequence. e.g.,

  msiSetRescSortScheme(random)
  msiSetRescSortScheme(byRescClass)

  will select randomly a cache class resource and put it on the top of the list.

This policy is distributed within the iRODS core.re file.

The rule is available at
https://github.com/irods/irods/blob/master/packaging/core.re.template

acSetRescSchemeForCreate {msiSetDefaultResc("demoResc","null");}
3 Data Management Policy Set (SILS LifeTime Library)

The LifeTime Library uses five additional policies to control creation of personal digital libraries for students. One of these policies modifies the option for selecting the default storage resource. A second policy turns on quota enforcement. Thus only three policies represent new rules. The policies are:

3.1 Turn on storage quota enforcement (Policy 10)
This rule implements restrictions on the total amount of storage space that can be used by a student. When the quota is exceeded, a student will be able to read files, but will not be able to write new files. The quota values are set by running the iodir command.

```
iodir suq UserName ResourceName - to set a quota on a storage resource
iodir suq UserName total     - to set a total storage quota
```

The rule is available at https://github.com/DICE-UNC/policyworkbook/blob/master/acRescQuotaPolicy.re

```
acRescQuotaPolicy {msiSetRescQuotaPolicy("on");}
```

3.1.1 Check for missing quotas
This policy identifies all accounts (user names) for which a quota has not been set. Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/sils-missing-quota.

```
main {
    # sils-missing-quota.r
    *rs = select USER_ID, USER_NAME;
    *Count = 0;
    foreach(*Row in *rs){
        *Usr = *Row.USER_ID;
        *Name = *Row.USER_NAME;
        # get quota
        *Q = select count(QUOTA_USER_ID) where QUOTA_USER_ID = *Usr;
        foreach(*R in *Q) {
            *User = *R.QUOTA_USER_ID;
            if(*User == "0") {
                writeLine("stdout", "No quota for *Name");
                *Count = *Count + 1;
            }
        }
    }
    writeLine("stdout", "Missing quotas for *Count users");
}
input null
output ruleExecOut
3.1.2 Calculate total storage usage

This policy calculates the total amount of storage used by person and identifies the person who has stored the most data. Updates to this policy are available from

```r
main {
  # sils-storageReport.r
  *rs = select USER_ID, USER_NAME;
  *Total = 0.0;
  *Max = 0.0;
  foreach(*Row in *rs){
    *Usr = *Row.USER_ID;
    *Name = *Row.USER_NAME;
    # get quota
    *Query1 = select sum(QUOTA_USAGE) where QUOTA_USAGE_USER_ID = *Usr;
    foreach(*Row1 in *Query1){
      *Use = double(*Row1.QUOTA_USAGE);
      if(*Use > 0.0) {
        *Gb = *Use/(1024.*1024.*1024.);
        writeLine("stdout", "Usage is *Gb Gbytes by *Name");
        *Total = *Total + *Use;
      }
      if(*Use >= *Max) {
        *Max = *Use;
        *Usem = *Name;
      }
    }
    writeLine("stdout", "Total usage is *Total bytes");
    writeLine("stdout", "Maximum usage is *Max bytes by *Usem");
  }
}
```

input null
output ruleExecOut

3.1.3 Identify persons who exceeded their quota

This rule identifies the individuals who have exceeded their quota and lists the top 10 users of storage. Updates to this policy are available from

This uses two policy functions, createList and addToList.

```r
main {
  # sils-checkQuota.r
  # Calculate storage quota for all users
  # Identify persons over their quota
  # List top *Num users
  *rs = select USER_ID, USER_NAME, USER_ZONE;
  *Total = 0.0;
  *Max = 0.0;
  *Numusers = 0;
  *Min = 0;
  createList("Listuse, *Num, "0.");
```
createList("Listnam, *Num, "");
writeLine("stdout","Name\tQuota\tUsage");
foreach(*Row in *rs){
  *Uid = *Row.USER_ID;
  *Name = *Row.USER_NAME;
  *Zone = *Row.USER_ZONE;
  *Numusers = *Numusers + 1;
  # get quota
  *Q = select QUOTA_OVER where QUOTA_USER_ID = *Uid;
  foreach(*R in *Q) {
    *Over = -double(*R.QUOTA_OVER)/1024./1024./1024.;
  }
  # get usage
  *Q2 = select sum(QUOTA_USAGE) where QUOTA_USAGE_USER_ID = *Uid;
  foreach(*R2 in *Q2) {
    *Usage = double(*R2.QUOTA_USAGE)/1024./1024./1024.;
    *Total = *Total + *Usage;
  }
  # find top *Num users
  if(*Usage > *Min) {
  }
  *Quota = (*Over + *Usage);
  *Usname = "*Name\t";
  if(strlen(*Name) < 8) {*Usname = "*Name\t"}
  writeLine("stdout","*Usname*Quota\t*Usage");
  if(*Over < 0.0) {
    writeLine("stdout","Quota *Quota exceeded by *Name, Usage is *Usage");
  }
  writeLine("stdout","Number of users is *Numusers");
  writeLine("stdout","Total usage is *Total Gbytes");
  writeLine("stdout","Top 10 users are");
  for (*I=0; *I<*Num; *I=*I+1) {
    *Us = elem(*Listnam,*I);
    *Usv = elem(*Listuse,*I);
    if(strlen(*Us) < 8) {*Us = "*Us\t"}
    writeLine("stdout","*Us\t*Usv");
  }
}
input *Num = 20
output ruleExecOut

3.1.4 Periodically update quota check

The storage usage is updated when the msiQuota micro-service is run. The usage can also be updated by running the administrative command: iadmin cu

This rule updates the usage every day. Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/sils-updateQuota.r.

checkQuota {
  # sils-updateQuota.r
  # rule to update usage daily
  delay("<PLUSET>1s</PLUSET><EF>1d</EF>") {
msiQuota;
writeLine("serverLog", "Updated quota check");
}
}
INPUT null
OUTPUT ruleExecOut

3.2 Manage selection of storage resource (Policy 11)
This rule changes the name of the default storage system that is used for storing files within the LifeTime Library.

Updates to the rule are available at
https://github.com/DICE-UNC/policy-workbook/blob/master/acSetRescSchemeForCreate.re

acSetRescSchemeForCreate {msiSetDefaultResc("lifelibResc1", "null");}

3.3 Manage selection of storage resource for replication (Policy 12)
This rule changes the default storage system name for replication of files within the LifeTime Library.

Updates to the rule are available at
https://github.com/DICE-UNC/policy-workbook/blob/master/acSetRescSchemeForRepl.re

acSetRescSchemeForRepl {msiSetDefaultResc("renci-unix1", "null");}

3.4 Enforce replication of each new file (Policy 13)
This rule implements an integrity requirement, ensuring that each file added to the LifeTime Library is replicated to a second storage system. The replication is queued for execution to minimize wait time on the original put action.

Currently, three post processing functions can be used individually or in sequence in the acPostProcForPut rule:

msiSysChksumDataObj – create a checksum on the file and store the checksum in the metadata catalog under the persistent state variable name "DATA_CHECKSUM".

msiExtractNaraMetadata - extract and register metadata from the just uploaded NARA files.

msiSysReplDataObj(replResc, flag) - can be used to replicate a copy of the file just uploaded or copied data object to the specified replica resource (replResc). Valid values for the "flag" input are "all", "updateRepl" and "rbudpTransfer". More than one flag values can be set using the "%" character as separator. e.g., "all%updateRepl". "updateRepl" means update an existing stale copy to the latest copy. The "all" flag means replicate to all resources in a resource group or update all stale copies if the "updateRepl" flag is also set. "rbudpTransfer" means the RBUDP protocol will be used for the transfer. A "null" input means a single replica will be made in one of the resources in the resource group. It may be desirable to do replication only if the dataObject is stored in a resource group.
Updates to the rule are available at
https://github.com/DICE-UNC/policy-workbook/blob/master/acPostProcForPut-ReplSILS.re

acPostProcForPut {ON($objPath like "/lifelibZone/home/*") {delay("<PLUSET>1s</PLUSET>")
{msiSysReplDataObj('renci-unix1','null');	}	}	}

3.5 Manage access control policy (Policy 14)
This rule keeps users from seeing the names of other user’s files, and is needed to ensure that each student collection is private to that student.

The rule sets the Access Control List policy. If the rule is not called or called with an argument other than STRICT, the STANDARD setting is in effect, which is fine for many sites. By default, users are allowed to see certain metadata, for example the data-object and sub-collection names in each other's collections. When access controls are made STRICT by calling msiAclPolicy(STRICT), the General Query Access Control is applied on collections and data object metadata which means that the list command, ils, will need 'read' access or better to the collection to return the collection contents (name of data-objects, sub-collections, etc.).

The default is the normal, non-strict level, allowing users to see names of other collections. In all cases, access control to the data-objects is enforced. Even if a person can see file names in a collection, “read” access is required on a file to be able to read the file. Even with STRICT access control, however, the admin user is not restricted so various microservices and queries will still be able to evaluate system-wide information. The session variable, "$userNameClient" can be used to limit actions to individual users. However, this is only secure in an irods-password environment (not GSI), but you can then have rules for specific users:

```
acAclPolicy {ON($userNameClient == "quickshare") { } }
acAclPolicy {msiAclPolicy("STRICT"); }
```
which was requested by ARCS (Sean Fleming). See rsGenQuery.c for more information on $userNameClient. The typical use is to just set it strict or not for all users.

Updates to the rule are available at
https://github.com/DICE-UNC/policy-workbook/blob/master/acAclPolicy-strict.re

```
acAclPolicy {msiAclPolicy("STRICT"); }
```
4 Data Administration Policy Set (RDA Practical Policy working group)

The Research Data Alliance Practical Policy working group conducted a survey of 41 sites that were managing data collections. A set of 11 policy categories that were applied across most of the sites was identified. The policies include automation of administrative functions, enforcement of management decisions, and validation of assessment criteria. The policies are listed in Table 1 and have minimal overlap with the policy sets for data sharing and student digital libraries, except for policies to manage access controls. For each policy category, multiple policies may be defined.

4.1 Data access control policies (Policy 14)

Automated application of access restrictions based on metadata simplifies administration of a data grid. Every repository needs to be able to easily restrict various data sets to specific audiences (e.g., campus members are granted read access due to licensing, while write access is granted to creators of a collection). This information is stored as system metadata and is checked on all accesses.

Access controls require the ability to assign a unique identifier to each person, validate the identity of each user, and then authorize each operation. Within the iRODS data grid, unique identifiers are assigned to users and files. The identifiers are used to associate access controls with a user name.

4.1.1 Find the User_ID associated with a User_name:

Since identifiers for users may be set as either strings (USER_NAME) or integers (USER_ID), a policy that allows a person to find the USER_ID for their USER_NAME is useful. This policy queries a metadata catalog and retrieves the USER_ID for the person who is running the rule. The policy can be applied interactively to files within a collection, or can be automated as part of a file ingestion process.

For the interactive version of the policy, the output is written to the screen. Updates to this policy are available from [http://github.com/DICE-UNC/policy-workbook/rda-userID.r](http://github.com/DICE-UNC/policy-workbook/rda-userID.r).

```r
myTestRule {
  # rda-userID.r
  # List information about the person running the rule
  *Query = select USER_ID where USER_NAME = '$userNameClient';
  foreach (*Row in *Query) {
    *userid = *Row.USER_ID;
    writeLine("stdout", "User: $userNameClient UserID: *userid");
  }
}
INPUT null
OUTPUT ruleExecOut
```
4.1.2 Find the File_ID associated with a file name:
Since identifiers for files may also be set as either strings (DATA_NAME) or integers (DATA_ID), a policy that finds the DATA_ID for a file is useful. This policy queries a metadata catalog, and retrieves the DATA_ID for a specified file name that is input to the rule. The result is written to the screen. The rule uses the policy functions:
- checkCollInput
- checkFileInput

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/rda-fileID.r.

```r
class myTestRule {  
  # rda-fileID.r  
  # find the DATA_ID associated with a file name  
  checkFileInput (~File);  
  ~Coll = "$rodsZoneClient/home/$userNameClient/" ++ ~RelativeCollectionName;  
  checkCollInput (~Coll);  
  ~Query = select DATA_ID where DATA_NAME = ~File and COLL_NAME = ~Coll;  
  foreach(~Row in ~Query) {  
    ~Dataid = ~Row.DATA_ID;  
    writeLine("stdout", "Collection ~Coll, File ~File, File ID ~Dataid");  
  }  
}  
INPUT ~File = 'foo1', ~RelativeCollectionName = 'test'  
OUTPUT ruleExecOut
```

4.1.3 Set write access control for a user:
A person can set an access control on a file that they own by specifying the file name, the desired access control, and the user name that will be given access. This policy reads as input the user name, the collection and file on which the access control is set, and the desired access control. The metadata catalog is updated to record the change in access control. This is similar to the ichmod command. This rule uses the policy functions:
- checkCollInput
- checkFileInput
- checkPathInput
- checkUserInput
- findZoneHostName

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/rda-setACL.r.

```r
class myTestRule {  
  # rda-setACL.r  
  # Input parameters are:  
  # Recursion flag  
  # default  
  # recursive - valid if access level is set to inherit  
  # Access Level
```
null
read
write
own
inherit
User name or group name who will have ACL changed
Path or file that will have ACL changed
checkUserInput(*User, $rodsZoneClient);
checkFileInput(*File);
*Home="/"$rodsZoneClient/home/$userNameClient/";
*Coll= *Home ++ *RelativeCollection;
checkCollInput(*Coll);
*Path = "*/File";
checkPathInput(*Path);
msiSetACL("default", *Acl,*User,*Path);
writeLine("stdout", "Set owner access for *User on file *Path");
}
INPUT *User="public", *RelativeCollection="test", *File="foo1", *Acl = "write"
OUTPUT ruleExecOut

4.1.4 Set operations that are allowable for the user "public"
This policy controls the operations that “public” users are allowed to execute. Only 2 operations are allowed - "read" - read files; and "query" - browse some system level metadata. Both operations can be specified by using the separator “%”. The rule uses the micro-service “msiSetPublicUserOpr” to specify what types of public access operations are allowed. The micro-services are called from a policy enforcement point associated with setting Public User Policy.

Update to the rule are available at
https://github.com/DICE-UNC/policy-workbook/blob/master/acSetPublicUserPolicy.re

acSetPublicUserPolicy {msiSetPublicUserOpr("read%query");}

4.1.5 Check the access controls on a file:
This policy checks each file in a collection for whether a specific user has access. This rule has input parameters for the names of a collection and user for which access controls will be checked. The desired access permission is compared with the access permissions set on the file. If the access control is not found, an error message is written. In practice, access control checks on files are enforced automatically by the iRODS framework. This rule uses policy functions:
checkCollInput
checkUserInput
findZoneHostName

Updates to this policy are available from
http://github.com/DICE-UNC/policy-workbook/rda-acl.r

IntegrityACL {
# rda-acl.r
#Rule to identify access controls on a file in a collection for a specific person
#Input
# Collection that will be analyzed
# Output
# List of files that the person can access
# Generate home collection name for admin running the rule
checkUserInput (*User, $rodsZoneClient);
*Path = "/$rodsZoneClient/home/$userNameClient/*Coll";
checkCollInput (*Path);

# Get USER_ID for the input user name
*Query = select USER_ID where USER_NAME = '*User';
*Userid = "";
foreach(*Row in *Query) {
  *Userid = *Row.USER_ID;
}

*Query3 = select DATA_ID, DATA_NAME, COLL_NAME where COLL_NAME like '*Path%';
foreach(*Row3 in *Query3) {
  *Dataid = *Row3.DATA_ID;
  *File = *Row3.DATA_NAME;
  *Col = *Row3.COLL_NAME;
  *Path = *Col ++ "/*File";
  writeLine("stdout","*Col/*File");
}

# Find ACL for the file
*Query4 = select DATA_ACCESS_TYPE, DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = '*Dataid';
foreach(*Row4 in *Query4) {
  *Userdid = *Row4.DATA_ACCESS_USER_ID;
  *Datatype = *Row4.DATA_ACCESS_TYPE;
  if(*Userid == *Userdid) {
    *Query5 = select TOKEN_NAME where TOKEN_NAMESPACE = 'access_type' and TOKEN_ID = '*Datatype';
    foreach(*Row5 in *Query5) {*Access = *Row5.TOKEN_NAME;}
    *Query6 = select USER_NAME where USER_ID = '*Userdid';
    foreach(*Row6 in *Query6) {*Usern = *Row6.USER_NAME;}
    writeLine("stdout"," User *Usern has access control *Access");
  }
}

INPUT *Coll =$"rules", *User =$"rwmoore"
OUTPUT ruleExecOut

4.2 Data format control policies (Policy 15)
Formats such as SPSS, SAS, and Stata will not be around forever so we need to move data out of such formats into open and more durable formats. Policies are needed to identify the data formats that are present in a collection, and transform obsolete data formats.
4.2.1 Set format conversion flag

A policy is needed to specify when format conversion is required. This policy sets a conversion flag when the data type is a specified format. The data type is normally defined for a file when it is loaded into the data grid. See the command

```bash
iput -D "data type" file-name
```

The rule uses the policy function:

```bash
checkCollInput
```

Updates to this policy are available from


---

```bash
listRule {
  # rda-setconv.r
  # Check files in a staging area
  # Set conversion flag "Conversion" to "ConvertMe" for specified filetypes
  # Check file types for Microsoft word
  # Input parameters
  # Relative path to staging area is *Collrel
  msiAddKeyVal(*Keyval,"Conversion","ConvertMe");
  # Generate full path name to staging area
  *Coll= "/$rodsZoneClient/home/$userNameClient/" ++ "*Collrel";
  checkCollInput(*Coll);
  *Q1 = select DATA_NAME where COLL_NAME = '*Coll';
  # Loop over files in the collection
  foreach(*R1 in *Q1) {
    *D = *R1.DATA_NAME;
    *Q2 = select DATA_TYPE_NAME where DATA_NAME = '*D' and COLL_NAME = '*Coll';
    *Data = *Coll ++ "/" ++ *D;
    # Set conversion flag for specified data types
    foreach(*R2 in *Q2) {
      *T = *R2.DATA_TYPE_NAME;
      if(*T == "*Type") {
        writeLine("stdout","Convert file *Data
        msiAssociateKeyValuePairsToObj(*Keyval,*Data,"-d");
      }
    }
  }
  INPUT *Type="Word format",*Collrel = "sub2"
  OUTPUT ruleExecOut
```

4.2.2 Invoke format conversion

This policy invokes the NCSA Polyglot service to transform a data format. This external service is invoked by sending http requests to a server at Drexel University. Note that the file that is being converted will also be moved to Drexel, with the converted file returned over the network. The rule uses the policy functions:

```bash
addAVUMetadata
deleteAVUMetadata
```

Updates to this policy are available from
The rule takes as input:
- **Aname** - flag with value "ConvertMe"
- **ItemName** - path of the file being converted

Output from the conversion program is:
- **out** - name of the converted file

```r
mytestrule(*Option,*ItemName,*AName,*AValue,*AUnit) {
    # rda-convertfile.r
    on(*AName == "ConvertMe") {
        *ItemName = "/$rodsZoneClient/home/$userNameClient/" ++ *ItemName;
        irods_curl_get("http://polyglot.cci.drexel.edu/", *ItemName, *AValue, *out);
        if(*out == "") {
            deleteAVUMetadata(*ItemName, "ConvertMe", *AValue, *AUnit, *out3);
            addAVUMetadata(*ItemName, "Conversion Error", *AValue, "dest", *out2);
        } else {
            addAVUMetadata(*out, "Derived from", *ItemName, "iRODS path", *out2);
            deleteAVUMetadata(*ItemName, "ConvertMe", *AValue, *AUnit, *out3);
        }
    }
}
```

**INPUT**
- *AName="ConvertMe",*ItemName="foo1.doc",*AValue="",*AUnit=""

**OUTPUT**
- ruleExecOut

### 4.2.3 Identify and archive specific file formats from a staging area

File format type is stored in a state information variable called **DATA_TYPE_NAME**. Queries can be issued against the metadata catalog to retrieve files with a given format type. Operations are also supported for extracting the file format type of a file, based on the file extension.

This policy examines a staging area for files with a specific format type. The file format is determined from the file extension. Files that have a desired extension, in this case an extension ".r", are moved into a specified collection. This makes it possible to sort files by file format type. The collection that corresponds to the staging area and the collection that corresponds to the destination archive are read from input. Note that when a file is moved, the access controls must be reset.

This rule uses the policy functions:
- checkCollInput
- createLogFile
- isColl

Updates to this policy are available from [http://github.com/DICE-UNC/policy-workbook/rda-stageformat.r](http://github.com/DICE-UNC/policy-workbook/rda-stageformat.r)
*Src = "/$rodsZoneClient/home/$userNameClient/" ++ *Stage;
*Dest= "/$rodsZoneClient/home/$userNameClient/" ++ *Coll;
checkCollInput (*Src);
checkCollInput (*Dest);
delay("<PLUSET>1m</PLUSET><EF>1d</EF>") {
  # Loop over files in a staging area, /$rodsZoneClient/home/$userNameClient/*stage
  # Put all files with .r into collection /$rodsZoneClient/home/$userNameClient/*Coll
  
  # ==== create a collection for log files if it does not exist

  #======== find files to stage
  *Query = select DATA_NAME where COLL_NAME = '"Src' and DATA_NAME like '%.r';
  foreach(*Row in *Query) {
    *File = *Row.DATANAME;
    *Src1 = *Src ++ "/" ++ *File;
    *Dest1 = *Dest ++ "/" ++ *File;
    #Check whether file already exists
    *Q3 = select count(DATA_NAME) where COLL_NAME = '"Dest' and DATA_NAME = '"File';
    foreach (*R3 in *Q3) { *DataID = *R3.DATA_NAME; }
    # Move file and set access permission
    if(*DataID == "0") {
      msiDataObjRename(*Src1,*Dest1, "0", *Status);
      msiSetACL("default","own",$userNameClient, *Dest1);
      writeLine(""*Lfile", "Moved file *Src1 to *Dest1";
    }
  }
}

INPUT *Stage=$"stage", *Coll=$"rules", *Res=$"demoResc"
OUTPUT ruleExecOut

4.3 Notification Policies (Policy 16)
Events that occur within the data management system can be logged in an audit trail. The audit trail can be parsed to analyze what has happened. Events can also be monitored, with appropriate E-mail sent to an administrator. Events can also be tracked through notifications that are sent to an indexing server each time a specified action occurs. Automated creation of event metadata is needed as data sets and data collections are being processed. Currently this is being done manually for most collections at great cost and effort.

4.3.1 Notify on collection deletion
Notification policies are implemented at Policy Enforcement Points, either before an action occurs or after the action is completed. A rule can be created that specifies the type of notification that will be used.

This policy sends E-mail to an administrator on deletion of a collection. A session variable, $collName, is used to identify which collection is being deleted.

Updates to the rule are available at
https://github.com/DICE-UNC/policy-workbook/blob/master/acPreProcForRmColl.re
4.3.2 Notification of events

Events can be detected at all policy enforcement points through use of a C++ version of the pluggable rule engine. The C++ version is fast enough to track all operations performed within the data management system. The detected events are documented in messages that are sent to a message queue for processing by an external indexing system. This capability will be available in version 4.2 of iRODS.

Policies can then be associated with each micro-service plugin to automate event detection and auditing. One application is the correlation of each change to the persistent state information with the event that caused the change. This requires mapping from client actions, to the policy enforcement points that are invoked, to the policies that are then enforced, to the micro-services that are executed, to the persistent state information attributes that are modified or changed. An example of how this can be done by hand is given in chapter 8. A similar approach can be used to audit all actions performed upon the data management system.

Computer actionable policies for monitoring events are listed in Chapter 5.6.

The “rule_exists” function tells the rule engine plugin system which rules this plugin listens to. In this case it listens to any rule under the "audit_" namespace.

The “exec_rule” function actually handles the auditing. It logs name, arguments, and the condInputData field of the REI in-memory structure of an operation, etc. to the server log.

We only show the relevant C++ code as the full code can be found on Github with the 4.2 release.

```cpp
#Declaring functions

acPreprocForRmColl {
    msiSendMail("admin@unc.edu", "Collection deletion", "Collection $collName is deleted");
}

irods::error rule_exists(irods::default_re_ctx& _ctx, std::string _rn, bool& _ret) {
    _ret = _rn.compare(0,6,"audit_") == 0;
    return SUCCESS();
}

irods::error exec_rule(irods::default_re_ctx& _ctx, std::string _rn, std::list<boost::any>& _ps, irods::callback _eff_hdlr) {
    std::stringstream ss;
    int i = 0;
    for (auto itr = begin(_ps); itr != end(_ps); ++itr) {
        if (itr->type() == typeid(std::string)) {
            ss << "arg" << i << "=" << boost::any_cast<std::string>(*itr) << std::endl;
        } else if (itr->type() == typeid(int)) {
            ss << "arg" << i << "=" << boost::any_cast<int>(*itr) << std::endl;
        } else {
        }
    }
    _eff_hdlr(_ctx, _rn, _ps, ss.str());
}
```
ss << "arg" << i << "= " << itr->type().name() << std::endl;
} 
 i++;
}
if (rei->condInputData) {
  for (i=0; i < rei->condInputData->len; i++) {
    ss << rei->condInputData->keyWord[i] << " = " << rei->condInputData->value[i] << std::endl;
  } 
} else {
  ss << "no condInputData" << std::endl;
}
return_eff_hdlr(std::string("writeLine"), std::string("serverLog"), ss.str());

4.4 Use agreement policies (Policy 17)
The creation of a use agreement requires an interaction with each user, independently of the data grid. The resulting information can be captured as metadata that is associated with each file in a collection. It is then possible to track whether a use agreement has been received, and write policies that restrict access when files have no official use agreement.

4.4.1 Set receipt of signed use agreement
A metadata attribute can be defined for each user to designate receipt of a signed user agreement. This is an example of a user-defined metadata attribute that can be associated with each user name.

The policy sets the use agreement for a specified user. This policy uses the metadata attribute “Use_Agreement” to store a value of “RECEIVED” when a use agreement is confirmed. The rule uses the policy function:

checkUserInput
findZoneHostName

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/rda-useSet.r.

ruleSetUse {
# rda-useSet.r
# Sets metadata attribute Use_Agreement to RECEIVED
checkUserInput (*User, $rodsZoneClient);
msiAddKeyVal(*Keyval, "Use_Agreement", "RECEIVED");
msiAssociateKeyValuePairsToObject(*Keyval,*User,"-u");
writeLine("stdout", "Set use agreement for *User");
}
INPUT *User = "rods"
OUTPUT ruleExecOut

4.4.2 Identify users without signed use agreement
This policy queries all user names to find users who either do not have a “Use_Agreement” metadata attribute name, or have a value that is not “RECEIVED”. If either case is found, a message is written to the screen.
Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/rda-useVerify.r.

```r
ruleCheckUse {
# rda-useVerify.r
# Checks whether the metadata attribute Use_Agreement has been set for each user
*Quser = select USER_NAME;
foreach (*Row in *Quser) {
  *User = *Row.USER_NAME;
  *Quse = select count(META_USER_ATTR_NAME) where USER_NAME = '*User' and META_USER_ATTR_NAME = 'Use_Agreement';
  foreach (*R1 in *Quse) {
    *Count = *R1.META_USER_ATTR_NAME;
    if (*Count != "0") {
      *Qcheck = select META_USER_ATTR_VALUE where USER_NAME = '*User' and META_USER_ATTR_NAME = 'Use_Agreement';
      foreach (*R2 in *Qcheck) {
        *Val = *R2.META_USER_ATTR_VALUE;
        if (*Val != "RECEIVED") {
          writeLine("stdout", "No use agreement for *User");
        }
      }
    }
  }
  else {
    writeLine("stdout", "No use agreement for *User");
  }
}
INPUT null
OUTPUT ruleExecOut
```

### 4.5 Integrity policy (Policy 18)

Policies are typically created to verify the integrity of files by comparing the current checksum with a saved value of the checksum. However, integrity policies can also be created to verify access controls on a collection, verify the presence of required metadata, verify file distribution, etc.

#### 4.5.1 Verify access controls on files

This rule analyses the files in a collection to verify that a required access control is present on each file. The input includes the name of the collection that will be verified, the type of access control that is required, and the name of a person for which the access control is set. The rule verifies the collection name, retrieves a USER_ID for the named person, and retrieves a DATA_ACCESS_DATA_ID number for the type of access control. A loop is made over the files in the collection, with a sub-loop that verifies the access control on each file. The results are printed to the screen. The rule uses the policy functions:

- `checkCollInput`
- `checkUserInput`
- `findZoneHostName`
integrityACL {
  # rda-integrityACL.r
  # Rule to analyze files in a collection
  # Verify that a specific ACL is present on each file in collection
  # Input
  # Collection that will be analyzed
  # Name of person to check for presence of ACL on file
  # Required ACL value, expressed as an integer
  # Output
  # Names of files that are missing the required ACL
  # Generate home collection name for user running the rule
  *Coll= "~/rodsZoneClient/home/$userNameClient/" ++ *Coll;

  # Verify input path is a collection
  checkCollInput (*Coll);
  checkUserInput (*User, $rodsZoneClient);
  # Get USER_ID for the input user name
  *Query = select USER_ID where USER_NAME = *User;
  *Userid = "";
  foreach(*Row in *Query) {
    *Userid = *Row.USER_ID;
  }

  # Get DATA_ACCESS_DATA_ID number that corresponds to requested access control
  *Query2 = select TOKEN_ID where TOKEN_NAMESPACE = 'access_type' and TOKEN_NAME = 'Acl';
  foreach(*Row2 in *Query2) {
    *Access = *Row2.TOKEN_ID;
  }
  writeLine("stdout", "Access control number of Acl is *Access");
  *Count = 0;

  # Loop over files in the collection
  *Query3 = select DATA_ID,DATA_NAME where COLL_NAME = *Coll';
  foreach(*Row3 in *Query3) {
    *Dataid = *Row3.DATA_ID;
    *File = *Row3.DATA_NAME;
    *Path = *Coll ++ "/" ++ *File
    # Find ACL for each file
    *Query4 = select DATA_ACCESS_TYPE, DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = *Dataid;

    # Loop over access controls for each file
    *Attrfound = 0;
    foreach(*Row4 in *Query4) {
      *Userdid = *Row4.DATA_ACCESS_USER_ID;
      if(*Userdid == *Userid) {
        *Attrfound = 1;
        *Datatype = *Row4.DATA_ACCESS_TYPE;
        if(*Datatype < *Access) {
          writeLine("stdout", "Path has wrong access permission, *Datatype");
        }
      }
    }
}

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/rda-integrityACL.r.
if(*Attrfound == 0) {
    writeLine("stdout", "*Path is missing access controls for *User");
    *Count = *Count + 1;
}
}
writeLine("stdout", "Number of files in *Coll missing access control for *User is *Count");
}
INPUT *Coll =$"rules", *User =$"rwmoore", *Acl = $"own"
OUTPUT ruleExecOut

4.5.2 Check integrity and number of replicas of files in a collection
This policy implements 17 basic operations needed for a production quality rule for verifying the integrity of a collection. The basic operations include:

1. Verifying all input parameters for consistency
2. Retrieving state information from the metadata catalog on each execution
3. Verifying integrity of each file by comparing the saved checksum with the computed checksum
4. Updating all replicas to the most recent version
5. Minimizing the load on the production services through a deadline scheduler
6. Differentiating between the logical name for the file and the physical location of the replicas
7. Identifying missing replicas and documenting their absence
8. Creating new replicas to replace missing files
9. Implementing load leveling to distribute files cross available storage systems
10. Creating a log file to record all repair operations and storing the log file in the data grid
11. Tracking progress of the policy execution
12. Initializing the rule for the first execution, including setting variables to track progress.
13. Enabling restart from the last checked file
14. Manipulating files in batches of 256 files at a time to handle arbitrarily large collections
15. Minimizing the number of sleep periods required by the deadline scheduler
16. Checking new files that have been added on a restart
17. Generating statistics about the execution rate and properties of the files that were checked.

Implementing all 17 operations increases the size of the production policy substantially. However, it is possible to show that the average time spent per file is still less than a disk rotation period, implying that the production rule is suitable for verifying integrity across arbitrarily large collections.

Each of these basic operations is annotated in the following integrity rule.

The policy to periodically check integrity uses the policy functions:

- addAVUMetadataToColl
periodicReplicaCreation {
  # rda-replication-rule.r
  # periodically check that required replicas are present, and create missing replicas
  checkCollInput (*Coll);
  checkRescInput (*Res,$rodsZoneClient);
  writeLine("stdout","Delay rule queued to periodically verify the number of replicas");
  delay("<PLUSET>1s</PLUSET><EF>7d</EF>") {
    # The replicas for each file are updated to the most recent version
    # Each file is checked to verify whether all required replicas exist and have valid checksums
    # As replicas are created, the algorithm round robs through available storage vaults
    # Checks that the number of storage resources used within a collection is greater than or
    # equal to the number of desired replicas.
    # This uses a just in time scheduler that slows down the processing rate
    # to complete the task within the specified number of seconds (*Delt)
    # Checks a TEST_DATA_ID parameter associated with the collection
    # to determine enable restarts after system interrupts
    # Writes a log file stored as Check-Timestamp in directory *Coll/log
    =======get current time, Timestamp is YYYY-MM-DD.hh:mm:ss =======
    msiGetSystemTime("TimeS","unix");
    *NumBadFiles = 0;
    *NumRepCreated = 0;
    *NumFiles = 0;
    *Runsize = double(0);
    *Sleeptime = 0;
    *colldataID = "0";
    #this is used to round robin through available storage resources
    *Jround = 0;
    ====== check whether the attribute TEST_DATA_ID has been set from a prior execution ======
    checkMetaExistsColl ("TEST_DATA_ID",*Coll,*Lfile,*colldataID);
    getNumSizeColl (*Coll,*colldataID,*Size,*Num);
    *= expected execution time = 0.0161 (sec) * (number of files) + (total size) / (50 MBytes/sec)
    =
    *Timeest = int(*Num / 62) + int(*Size / 50000000);
    writeLine("*Lfile","Estimated time is *Timeest seconds, total time is *Delt seconds, number

of files is *Num, and total size is *Size bytes*;  
writeLine("*Lfile","Number of required copies of a file is *NumReplicas");  
if(*Delt > 0 && *Size > 0) {  
  *Fac = *Size / *Delt;  
  writeLine("*Lfile","Required analysis rate is *Fac bytes/second");

#============= identify the resources that were used for the collection =============

getRescColl (*Coll, *Rlist, *Ulist0, *Lfile, *Ir);  
*Irm1 = *Ir - 1;  
if(*Ir < *NumReplicas) {  
  writeLine("*Lfile","Required number of replicas, *NumReplicas, exceeds the number of  
storage vaults, *Ir");  
  fail;  
}  
}  

#========== loop over all the files in the collection in batches of 256 ===========

*iter = 0;  
*q2 = select order(DATA_ID), DATA_SIZE, DATA_NAME, COLL_NAME, DATA_CHECKSUM  
where COLL_NAME like *'Coll%' and DATA_ID > *colldataID';  
foreach(*r2 in *q2) {  
  *iter = *iter + 1;  
  *Sizedata = *r2.DATA_SIZE;  
  *newdataID = *r2.DATA_ID;  
  *Name = *r2.DATA_NAME;  
  *Colln = *r2.COLL_NAME;  
  if(*Colln != *LPath) {  
    #======= before updating replicas, must verify that the replica has a valid checksum ========
*NumBadFiles);  
    #========== pick resource to use as source ========================================
    selectRescUpdate (*Rlist, *Ulist, *Ir, *Resource);  
    msiDataObjRepl("*Colln/*Name","updateRepl=++++rescName=*Resource",*Status2);  
    if(*Status2 != 0) {  
      writeLine("*Lfile","Unable to update replicas to most recent version for *Colln/*Name");  
    }  
  }  
  # end of error message if not able to update replicas to most recent version  
  #======== test whether the required number of replicas exists ================  
  if (*Numr != *NumReplicas) {  
    *N = *NumReplicas - *Numr;  
*NumRepCreated);  
  }  
  # end of check that the required number of replicas is not present  
  #====== slow rate at which are processing collection to meet deadline =============
  *Runsize = *Runsize + double(*Sizedata);  
  msiGetSystemTime(*timei, "unix");  
  *timerun = int(*TimeS) + *Runsize / *Fac;  
  *delt = *timerun - int(*timei);  
  if (*delt > 4) {  
    msiSleep(str(*delt), "0");  
    writeLine("*Lfile","Sleeping for *delt");  
  }  
  # end of check on length of sleep time  
  *NumFiles = *NumFiles + 1;  
  if (*iter % 256 == 0) {  
    updateCollMeta (*Coll, "TEST_DATA_ID", *colldataID, *newdataID, *Lfile);  
  }  
  # end of test to update TEST_DATA_ID  
}  
# end of iteration check  
}  
# end of loop over files  
writeLine("*Lfile","Number of logical file names tested is *NumFiles, total size checked is
*Runsize bytes, and total time slept is *Sleeptime seconds”;

    writeLine("*Lfile", "Number of bad files is *NumBadFiles, and number of replicated files
created is *NumRepCreated");

#====== reset TEST_DATA_ID status flag to zero =====================

    *Query6 = select META_COLL_ATTR_VALUE where COLL_NAME = '*Coll' and
META_COLL_ATTR_NAME = 'TEST_DATA_ID';

    foreach(*Row6 in *Query6) {
        *colldataID = *Row6.META_COLL_ATTR_VALUE;
    } # end of loop to get *colldataID

    updateCollMeta (*Coll, "TEST_DATA_ID", *colldataID, "0", *Lfile);

} # end of check on evaluation bandwidth

#====== Calculate actual elapsed time ==============================

    msiGetSystemTime("*TimeE", "unix");

    *Del = int(*TimeE) - int("*TimeS");

    writeLine("*Lfile", "Total elapsed time is *Del seconds");

} # end of delay command

INPUT *Coll="/testZone/home/rwmoore/test", *Delt=2, *NumReplicas = 2,
*Res="demoResc"

OUTPUT ruleExecOut

4.6 Metadata extraction (Policy 19)

The necessary task in building a digital library is the creation of provenance and
descriptive metadata. This typically requires interactive creation of the descriptive
metadata. For collections that have more than a thousand digital objects, this
becomes a laborious task. If the metadata attributes can be aggregated into a
standard format, then bulk loading of metadata may be appropriate. Examples
include bulk loading from an XML file or a pipe-delimited file.

An alternate approach is “feature-based” indexing, in which the digital object is
examined for the presence of desired features. Information about a feature is
extracted and registered as metadata on the digital object. An example is pattern-
based recognition of descriptive metadata within a text file.

4.6.1 Load metadata from an XML file

Metadata can be loaded into a data grid directly from an XML file. This policy
assumes a specific structure for the XML file of the form:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
    <AVU>
        <Target>/rodsZoneClient/home/$userNameClient/XML/sample.xml</Target>
        <Attribute>Order ID</Attribute>
        <Value>889923</Value>
        <Unit />
    </AVU>
    <AVU>
        <Target>/rodsZoneClient/home/$userNameClient/XML/sample.xml</Target>
        <Attribute>Order Person</Attribute>
        <Value>John Smith</Value>
        <Unit />
    </AVU>
</metadata>
```
Note that this specifies the target file to which the metadata is added. Each metadata attribute, value, and unit is formed into an AVU that is attached as metadata to the file. The rule uses the policy function:

```
cHECKPATHINPUT
```

Updates to this policy are available from [http://github.com/DICE-UNC/policy-workbook/rda-loadMetadataFromXml.r](http://github.com/DICE-UNC/policy-workbook/rda-loadMetadataFromXml.r).

```r
myTestRule {
  # rda-loadMetadataFromXml.r
  # Input parameters
  # targetObj - iRODS target file that metadata will be attached to, null if Target is specified
  # xmlObj - iRODS path to XML file that metadata is drawn from
  # xmlObj is assumed to be in AVU-format
  #
  # This format is created by transforming the original XML file
  # using an appropriate style sheet as shown in rulemsiXsltApply.r
  # This micro-service requires libxml2.

  *targetObj = "/$rodsZoneClient/home/$userNameClient/" ++ *targetObj;
  *xmlObj = "/$rodsZoneClient/home/$userNameClient/" ++ *xmlObj;
  if (*targetObj != "") {CHECKPATHINPUT (*targetObj);}
  CHECKPATHINPUT (*xmlObj);
  msiLoadMetadataFromXml(*targetObj, *xmlObj);
  write message to stdout
  WRITELINE("stdout","Extracted metadata from *xmlObj and attached to *targetObj");
}
```

**4.6.2 Load metadata from a pipe-delimited file**

Metadata can be loaded into a data grid directly from a pipe-delimited file. This policy assumes a specific structure for the pipe-delimited file of the form:

```
File-name |attribute-name |attribute-value
File-name |attribute-name |attribute-value |units
C-collection-name |attribute-name |attribute-value
C-collection-name |attribute-name |attribute-value |units
```

For the specified File-name or collection-name, the pipe-delimited values for the attribute name, the attribute value, and the attribute units or comments can be bulk loaded. This rule uses the policy function:

```
CHECKPATHINPUT
```

Updates to this policy are available from [http://github.com/DICE-UNC/policy-workbook/rda-metaloadpipe.r](http://github.com/DICE-UNC/policy-workbook/rda-metaloadpipe.r).

```r
myTestRule {
  # rda-metaloadpipe.r
  # Input parameter is:
```
# Path name of file containing metadata
# Format of file is
# C-collection-name | Attribute | Value | Units
# Path-name-for-file | Attribute | Value

Example

# /lifelibZone/home/rwmoore/foo1 | Test | 34

#Example

#Output parameter is:
# Status

"*Path="/rdsZoneClient/home/$userNameClient/" ++ *Coll;
checkPathInput (*Path);
msiLoadMetadataFromDataObj(*Path,*Status);
writeLine("stdout","Loaded metadata from file *Path");
}

INPUT *Coll=$"Rules/metapipe"

OUTPUT ruleExecOut

4.6.3  Contextual metadata extraction through pattern recognition

Pattern matching operations can be applied to text to extract contextual metadata. A template for pattern matching can be created that defines triplets:

<pre-string-regexp, keyword, post-string-regexp>

The triplets are read into memory, and then used to search a data buffer. For each set of pre and post regular expressions, the string between them is associated with the specified keyword and can be stored as a metadata attribute on the file.

In the example, the template file has the format:

<pre-tag>X-Mailer: </pre-tag>Mailer User<pre-tag>
</post-tag>

<pre-tag>Date: </pre-tag>Sent Date<pre-tag>
</post-tag>

<pre-tag>From: </pre-tag>Sender<pre-tag>
</post-tag>

<pre-tag>To: </pre-tag>Primary Recipient<pre-tag>
</post-tag>

<pre-tag>Cc: </pre-tag>Other Recipient<pre-tag>
</post-tag>

<pre-tag>Subject: </pre-tag>Subject<pre-tag>
</post-tag>

<pre-tag>Content-Type: </pre-tag>Content Type<pre-tag>
</post-tag>

The end tag is actually a "return" for unix systems, or a "carriage-return/line feed" for Windows systems. The example rule reads a text file into a buffer in memory, reads in the template file that defines the regular expressions, and then parses the text in the buffer to identify presence of a desired metadata attribute. The rule uses the policy function:

checkPathInput

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/rda-metaload.r.
myTestRule {
  # rda-metaload.r
  # Input parameters are:
  # Buffer
  # Tag structure
  # Output parameter is:
  # Keyval pair buffer
  "Tag = "/$rodsZoneClient/home/$userNameClient/" ++ *Tag;
  "Pathfile = "/$rodsZoneClient/home/$userNameClient/" ++ *Pathfile;
  "Outfile = "/$rodsZoneClient/home/$userNameClient/" ++ *Outfile;
  checkPathInput (*Tag);
  checkPathInput (*Outfile);
  checkPathInput (*Outfile);

  # Read in 10,000 bytes of the file
  msiDataObjOpen(*Pathfile,*F_desc);
  msiDataObjRead(*F_desc,*Len,*File_buf);
  msiDataObjClose(*F_desc,*Status);

  # Read in the tag template file
  msiDataObjOpen(*Tag,*T_desc);
  msiDataObjRead(*T_desc, 10000, *Tag_buf);
  msiReadMDTemplateIntoTagStruct(*Tag_buf,*Tags);
  msiDataObjClose(*T_desc,*Status);

  # Extract metadata from file using the tag template file
  msiExtractTemplateMDFromBuf(*File_buf,*Tags,*Keyval);

  # Write out extracted metadata
  writeKeyValPairs("stdout",*Keyval," : ");
  msiGetObjType(*Outfile,*Otype);

  # Add metadata to the object
  msiAssociateKeyValuePairsToObj(*Keyval,*Outfile,*Otype);
}

INPUT *Tag="Rules/email.tag", *Pathfile="Rules/sample.email",
*Outfile="Rules/sample.email", *Len=10000
OUTPUT ruleExecOut

4.6.4 Stripping metadata from a file
It may be necessary to strip metadata from a file before adding the required metadata. The following rule takes as input the path to the file, and removes descriptive metadata. The rule uses the policy function:
checkPathInput

Updates to this policy are available from
4.7 Data backup policies (Policy 20)

Data backup can take multiple forms:

- Time-stamped copies of digital objects that are saved in a separate collection
- Replicas of digital objects that can be accessed when the original is unavailable
- Copies of digital objects that are put into separate collections or data grids

The choice depends upon whether a time history of the evolution of the file is needed or whether recovery is needed when files are corrupted.

4.7.1 Data versioning policy

A version of a file can be created by adding a time stamp, and moving the version to an archive directory. This rule processes files in a collection, creating a version of each file that is stored in a destination directory called "SaveVersions". The rule uses the policy function:

```
checkCollInput
```

Updates to this policy are available from

http://github.com/DICE-UNC/policy-workbook/rda-version.r
The version number can be inserted in the file name before the extension. This rule parses the file name, identifies an extension, and inserts the time stamp before the extension when the version name is created. The rule uses the policy function:

cHECKPathInput

Updates to this policy are available from
4.7.2 Data backup staging policy

Within the iRODS data grid, backups, copies, and replicas can be supported. The difference is the set of state information that is needed for each type of entity. A backup is a time-stamped copy of a file. A replica is an additional copy of a file that is stored on a separate storage system. The replica number is tracked along with whether the original has been changed. Generic state information includes a creation time for the data object, the location where the data object is stored, the owner of the data object, modification time stamps, and access controls. An outcome of this approach is that it is possible to use the same client to access backups, copies, and replicas.

This rule creates a time-stamped backup directory, and copies all of the files from the source directory to the backup directory. The rule reads from input the collection for which the backup will be done, the storage location where the backups will be stored, and the destination collection that will hold the backup. Within the destination collection, a time-stamped sub-directory is created to hold each backup set. The rule checks the input, checks that each operation completes correctly, and writes information to a server log. The rule uses the policy function: checkCollInput

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/rda-backup.r.

```
myTestRule {
  # rda-backup.r
  # Test delayed execution
  "Home = "/$rodsZoneClient/home/$userNameClient/";
  "Source = "Home ++ "Collrel;"
  checkCollInput (*Coll);
  "Dest = "Home ++ "Destrel;"
  checkCollInput (*Dest);
  writeLine("stdout", "Backup collection "Source");
  delay("<PLUSET>1s</PLUSET><EF>7ds</EF>") {
    msiGetSystemTime("Time", "human");
    # Create backup collection with name *Dest/Check-Timestamp
    #======================get current time, Timestamp is YYYY-MM-DD.hh:mm:ss
    msiGetSystemTime("TimeH", "human");
    #====================== create a collection for backup if it does not exist ========
    "Lpath = "Dest ++ "/" ++ "TimeH;
    "Q1 = select count(COLL_NAME) where COLL_NAME = "Lpath;"
    foreach(*R1 in *Q1) {
      "Result = *R1.COLL_NAME;
    }
    if(*Result == "0") {
      msiCollCreate(*Lpath, "1", *Status);
      if(*Status < 0) {
        writeLine("serverlog", "Could not create backup collection");
        fail;
      }
    }
  }
}  # end of check on status
```
4.7.3 Copy files to a federated staging area

This rule takes all files in a "stage" directory on the first data grid, copies them to an "Archive" directory on the second data grid, and deletes the file from the first data grid. The rule also logs all of the actions and writes the log to a directory in the second data grid. The rule uses the policy functions:

- checkCollInput
- checkRescInput
- createLogFile
- findZoneHostName
- isColl

Updates to this policy are available from
http://github.com/DICE-UNC/policy-workbook/rda-stage.r.

```
# Loops over files in staging area
# Put all files into collection
# Move file and set access permission
# verify checksum
```

# Delete file from staging area if checksum is good
else {
    msiDataObjUnlink("objPath=*Src1+++forceFlag=", *Status);
}
}
}
INPUT *Stage =$"stage", *Coll=$"Archive", *DestZone=$"tempZone", *Res=$"demoResc"
OUTPUT ruleExecOut

4.8 Data retention policies (Policy 21)
Each file in a collection may have a different retention period, or all files in a collection may have the same retention period. The iRODS data grid specifies a data expiration date in the metadata attribute “DATA_EXPIRY”. The expiration date is stored as a Unix time variable. Information about the creation time of each file is stored in the metadata attribute DATA_CREATE_TIME.

4.8.1 Purge policy to free storage space
This policy manages a cache to ensure that a minimum amount of free space is available for deposition of new files. The policy runs periodically, every 24 hours. An information catalog is queried to find the total amount of storage space that is being used. This is compared to an input parameter that specifies the maximum allowed space. Additional input parameters specify the collection and the storage resource names. A second query retrieves information about the file names, file sizes, and creation time. The result set is ordered by the creation date, making it possible to loop over the files, deleting the oldest files until the required free space is available.

This policy was developed by Jean-Yves Nief of the French National Institute for Nuclear Physics and Particle Physics Computer Center. This rule could be modified to purge old backup directories. The rule uses the policy functions:
checkCollInput
checkRescInput
findZoneHostName

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/rda-purge.r.

purgeDiskCache {
# rda-purge.r
checkRescInput (*CacheRescName, $rodsZoneClient);
checkCollInput (*Collection);
delay("<PLUSET>30s</PLUSET><EF>24h</EF>") {
    *Q1 = select sum(DATA_SIZE) where RESC_NAME = '*CacheRescName';
    foreach (*R1 in *Q1) {
        *TotalSize = *R1.DATA_SIZE;
    }
    *usedSpace = double(*TotalSize);
    *MaxSpAlwd = *MaxSpAlwdTBs * 1024^4;
    if ( *usedSpace > *MaxSpAlwd ) then {
        msiGetIcatTime("Time", "unix");
    }
}
*Q2 = select DATA_NAME, COLL_NAME, DATA_SIZE, order(DATA_CREATE_TIME) where DATA_RESC_NAME = "CacheRescName" AND COLL_NAME like '*Collection%';
foreach (*R2 in *Q2) {
    *D = *R2.DATANAME;
    *C = *R2.COLL_NAME;
    *S = *R2.DATASIZE;
    *usedSpace = *usedSpace - double(*S);
    if (*usedSpace < *MaxSpAlwd) {
        break;
    }
    msiDataObjTrim("C/*D","CacheRescName",null,1,1,status);
    writeLine("stdout","C/*D on CacheRescName has been purged");
}
}
}
}
}
}
input *MaxSpAlwdTBs = $1, *Collection = "/tempZone", *CacheRescName = "demoResc"
output ruleExecOut

4.8.2 Data expiration policy
This policy checks the date specified by an expiration metadata attribute that has been assigned to the file, and creates a list of all files that have expired. Input parameters are used to specify the collection that is being checked and whether expired files should be found. A query is made to the information catalog to get a list of the DATA_EXPIRY date for each file. This is compared to the current Unix time. Files that have expired are listed and the total number is counted. The rule uses the policy function:
checkCollInput

Updates to this policy are available from
*File = *R1.DATA_NAME;
writeLine("stdout", "File *File has not expired");
*Count = *Count + 1;
}
if(*Attrname <= *Time && *Flag == "EXPIRED") {
*File = *R1.DATA_NAME;
writeLine("stdout", "File *File has expired");
*Counte = *Counte + 1;
}
}
if(*Flag == "EXPIRED") {writeLine("stdout", "Number of files in *Coll that have expired is *Counte");
if(*Flag == "NOT EXPIRED") {writeLine("stdout", "Number of files in *Coll that have not expired is *Count");
}
INPUT *Coll = "/$rodsZoneClient/home/$userNameClient/sub1", *Flag = "EXPIRED"
OUTPUT ruleExecOut

4.9 Disposition policy for expired files (Policy 22)
Files in the iRODS data grid can be tagged with additional metadata attributes. For example, a metadata attribute with the name "Retention_Flag" can be added to each file, along with a metadata attribute value such as “EXPIRED” or “NOT_EXPIRED”. By using metadata to track the status of each file, it is possible to separate the retention policy from the disposition policy. The retention policy can set the metadata attribute, and the disposition policy can read the metadata attribute.

This rule migrates files to an archive that have a metadata attribute with the name “Retention_Flag” that has the value "EXPIRED”. The rule reads as input the name of the collection that will be checked and the name of the destination collection. The collection names are verified. A query is then issued to the information catalog to retrieve the names of the files in the collection that have the “EXPIRED” value for the “Retention_Flag”. All of the returned files in the list are moved to the destination collection. Note that the access controls on the file will need to be reset after the move. The rule uses the policy function:
   checkCollInput

Updates to this policy are available from http://github.com/DICE‐UNC/policy‐workbook/rda‐disposition.r.

rule disposition {
  # rda-disposition.r
  # Input parameter is:
  # Name of collection that will be checked
  # Retention_Flag with value "EXPIRED" or "NOT_EXPIRED"
  # Output is:
  # Migration of "EXPIRED" files to an archive collection
  *Coll = "/$rodsZoneClient/home/$userNameClient/" ++ *Collrel;
  *Dest = "/$rodsZoneClient/home/$userNameClient/" ++ *Archiverel;
  # Verify that input path is a collection
  checkCollInput (*Coll);
  # Verify that archive path is a collection
checkCollInput(*Dest);
*Count = 0;
# Loop over files in the collection
*Q1 = select DATA_ID,DATA_NAME where COLL_NAME = '*'Coll' and
META_DATA_ATTR_NAME = 'Retention_Flag' and META_DATA_ATTR_VALUE = 'EXPIRED';
foreach(*R1 in *Q1) {
  *File = *R1.DATA_NAME;
  # move the file to the archive
  *SourceFile = *Coll ++ "/" ++ *File;
  *DestFile = *Dest ++ "/" ++ *File;
  msiDataObjRename(*SourceFile,*DestFile,"0",*Status);
  if (*Status < 0) {
    writeLine("stdout", "File *SourceFile could not be archived");
  } else { *Count = *Count + 1; }
} WRITE_line("stdout", "Migrated *Count files to the archive *Dest");

4.10 Restricted searching policy (Policy 23)
Search policies may be applied to the names of files, or to the descriptive metadata, or to system state information. A data grid administrator may be able to examine all of the metadata and see all file names, but an individual user may only be able to see the content that they own. A new genquery interface is being developed for iRODS version 4.2 which will support access controls on metadata.

4.10.1 Strict access control
The most commonly requested restriction is to limit the ability of users to see any other user’s files. This can be applied to all users, or applied to a specific user.

A strict access control is implemented through the Policy Enforcement Point called acAclPolicy. The micro-service msiAclPolicy implements the restriction.

Updates to the rule are available at
https://github.com/DICE-UNC/policy-workbook/blob/master/acAclPolicy-strict.re

    acAclPolicy {msiAclPolicy("STRIC"); }

4.10.2 Controlled queries
A query to an external database can be created and registered as a database object. CLICKing on the registered query will cause the query to be executed with the results returned as a file. This makes it possible to control interactions with search engines.

4.11 Storage cost reports (Policy 24)
Reports can be generated that summarize the use of any aspect of the data grid. The most common reports detail usage by user by storage system.

4.11.1 Usage report by user name and storage system
The basic approach is to calculate the amount of storage used on each storage device and then to generate a cost by multiplying usage by the charge per storage
for the device type. This can be refined to implement a separate cost per storage device. The cost information can be stored as a metadata attribute that is associated with each storage resource.

This rule sums the amount of storage used for each device by each user. A query is issued to the information catalog that sums the storage for each home directory in the data grid. The result is written to the screen.

Updates to this policy are available from [http://github.com/DICE-UNC/policy-workbook/rda-storage.r](http://github.com/DICE-UNC/policy-workbook/rda-storage.r).

```r
ruleStorage {
  # rda-storage.r
  # Total the size of files on each storage vault for each user
  # Get list of users
  *Quser = select USER_NAME;
  *Totsize = 0.0;
  *Totnum = 0;
  *Numusers = 0;
  for (*Row in *Quser) {
    *Numusers = *Numusers + 1;
    *User = *Row.USER_NAME;
    *Path = "/$rodsZoneClient/home/*User/%";
    *Q = select sum(DATA_SIZE),count(DATA_ID), DATA_RESC_NAME where COLL_NAME like 
      '*Path';
    for (*R in *Q) {
      *Size = double(*R.DATA_SIZE)/1024./1024./1024.;
      *Num = *R.DATA_ID;
      *V = *R.DATA_RESC_NAME;
      if(*Size > 0.) {
        writeLine(“stdout”, “ Storage on *V is *Size GBytes, Number of files is *Num, *User”);
        *Totsize = *Totsize + *Size;
        *Totnum = *Totnum + int(*Num);
      }
    }
  }
  writeLine("stdout", "Total storage is *Totsize GBytes for *Totnum files for *Numusers");
}
INPUT null
OUTPUT ruleExecOut
```

### 4.11.2 Cost report by user name and storage system

A cost algorithm is implemented by storing a “cost per byte” metadata attribute on each storage resource. The “cost per byte” attribute is stored as the metadata attribute called “Storage_Cost”, with the attribute value equal to the storage cost per byte. A query is issued to the information catalog to get a list of the users. Then for each user, a query is issued to sum the storage for each user for each storage device. The storage cost per byte is retrieved by a query, and the storage cost is calculated.

Updates to this policy are available from
ruleStorage {
  # rda-storageCost.r
  # Total the cost for files on each storage vault for each user
  # Get list of users
  *Quser = select USER_NAME;
  foreach (*Row in *Quser) {
    *User = *Row.USER_NAME;
    writeLine("stdout", "Storage for *User");
    *Path = "/$rodsZoneClient/home/*User/%";
    *Q = select sum(DATA_SIZE),DATA_RESC_NAME where COLL_NAME like "*Path";
    foreach (*R in *Q) {
      *Size = *R.DATA_SIZE;
      *V = *R.DATA_RESC_NAME;
      *Qresc = select META_RESC_ATTR_VALUE where RESC_NAME = "*V" and META_RESC_ATTR_NAME = 'Storage_Cost';
      foreach (*Rowc in *Qresc) {
        *Cost = int(*Rowc.META_RESC_ATTR_VALUE);
      }
      *Scost = *Cost * int(*Size);
      writeLine ("stdout", " Storage cost on *V is *Scost");
    }
  }
}

INPUT null
OUTPUT ruleExecOut
5 Odum Data Preservation Policy set

The preservation policies overlap with the RDA data management policies. Table 1 shows how the policy sets are related. The Odum data preservation policies typically required integration with additional software systems for implementation. Thus:

- De-identification of data
- Applying unique data identifiers
- Data normalization to non-proprietary formats
- Authentication identity management
- Creation of PREMIS event data
- Assessment criteria validation
- Mapping metadata across systems
- Automatic checksums
- Tracking use

<table>
<thead>
<tr>
<th>Process</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>De-identification of data</td>
<td>Uses Bitcurator</td>
</tr>
<tr>
<td>Applying unique data identifiers</td>
<td>Uses Handle system</td>
</tr>
<tr>
<td>Data normalization to non-proprietary formats</td>
<td>Uses Polyglot</td>
</tr>
<tr>
<td>Authentication identity management</td>
<td>Uses InCommon</td>
</tr>
<tr>
<td>Creation of PREMIS event data</td>
<td>Uses message bus</td>
</tr>
<tr>
<td>Assessment criteria validation</td>
<td>Uses indexing technology</td>
</tr>
<tr>
<td>Mapping metadata across systems</td>
<td>Uses HIVE</td>
</tr>
<tr>
<td>Automatic checksums</td>
<td>Uses SHA-128</td>
</tr>
<tr>
<td>Tracking use</td>
<td>Uses DataBook</td>
</tr>
</tbody>
</table>

5.1 Automate access restrictions (Policy 14)

One approach is to associate access restrictions with a collection, and then have all files within the collection inherit the access controls. When a file is put into the collection, the required access controls are automatically applied.

5.1.1 Set inheritance of access controls on a collection

Access controls on a file can be inherited from the collection into which the file is organized. This rule reads as input the collection name and then sets an “inherit” flag on the collection. Files that are deposited into the collection will “inherit” the access controls that were set on the collection. The rule uses the policy function:

```
checkCollInput
```

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/odum-inherit.r.

```
setInheritance {
  # odum-inherit.r
  # Input parameters are:
  # Recursion flag
  # default
  # recursive - valid if access level is set to inherit
  # Access Level
  # null
  # read
  # write
  # own
  # inherit
  # User name or group name who will have ACL changed
  # Path or file that will have ACL changed
  *Home"="/sandboxClient/home/$userNameClient/"
  *Path= *Home ++ *RelativeCollection;
  checkCollInput (*Path);
  msiSetACL("recursive", *Acl,*User,*Path);
```
5.1.2 Check whether a specific person has access to a collection
The rule shown in section 4.1.5 checks each file in a collection to determine whether a specified person has access. The type of access control is displayed. The rule finds the person’s USER_ID and the DATA_ID for each file in the collection.

5.1.3 Identify all persons with access to files in a collection
This rule creates a list of all of the persons who have access to any file within a collection. The number of files that can be accessed and the total size of the accessible files is calculated. The rule uses the policy function:

```
listUserAccess {
# odum-list-ACL.r
  *c = "/$rodsZoneClient/home/$userNameClient";
  summary(*c);
} summary (*c) {
  # rule to list all persons who have access to a home collection
  #Get USER_ID for the input user name
  *Query = select USER_ID where USER_NAME = '$userNameClient';
  *Userid = "";
  foreach(*Row in *Query) {
    *Userid = *Row.USER_ID;
  }
  if(*Userid == "") {
    writeLine("stdout", "Input user name *User is unknown");
    fail;
  } else {writeLine("stdout", "UserID is *Userid");}
  # loop over files in home collection
  *Coll = "*c%";
  *rs = select DATA_ID, DATA_SIZE where COLL_NAME like "*Coll";
  *res.total = str(0);
  *total.total = str(0);
  foreach(*r in *rs) {
    *fn = *r.DATA_ID;
    *ds = *r.DATA_SIZE;
    # Find ACL for the file
    *Query4 = select DATA_ACCESS_TYPE, DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = "*fn";
    # Loop over access controls for each file
    foreach(*Row4 in *Query4) {
      *Userdid = *Row4.DATA_ACCESS_USER_ID;
      *Datatype = *Row4.DATA_ACCESS_TYPE;
    }
  }
}
if(*Userid != *Userdid) {
    *Query5 = select TOKEN_NAME where TOKEN_NAMESPACE = 'access_type' and TOKEN_ID = *Datatype;'
    foreach (*Row5 in *Query5) {*Access = *Row5.TOKEN_NAME;}
    *Query6 = select USER_NAME where USER_ID = *Userdid';
    foreach (*Row6 in *Query6) {*Usern = *Row6.USER_NAME;}
    # *DATA_ID has access control *Access for user *Usern
    # Count number of files accessible by this user, and size of files accessible by user
    if (contains(*res, *Usern)) {
        *res.*Usern = str(int(*res.*Usern) + 1)
        *total.*Usern = str(double(*total.*Usern) + double(*ds))
    } else {
        *res.*Usern = str(1);
        *total.*Usern = *ds;
    }
    *res.total = str(int(*res.total) + 1);
    *total.total = str(double(*total.total) + double(*ds));
}
writeLine("stdout", "usern\ttcount\tavg\tsize\ttotal\tsize");
foreach(*Usern in *res) {
    *Us = "\t*Usern\t";
    if(strlen(*Usern) >= 8) {*Us = "\t*Usern\t";}
    if(*Usern != "total") {
        writeLine("stdout", "*Us"++*res.*Usern++"\t\t"++str(double(*total.*Usern)/int(*res.*Usern))++"\t\t"++*total.*Usern);
    }
    writeLine("stdout", "total\n\t++*res.total++"\t++str(double(*total.total)/int(*res.total))++"\t++*total.total",
    input null
    output ruleExecOut

5.1.4 Identify files that can be accessed by an account

Once a collection has been analyzed to determine which accounts have access, the list of account names can be examined to determine which account access should be deleted. The following rule lists all of the files that can be accessed by a specified account. The rule uses the policy function:

    checkUserInput
    findZoneHostName

    identifyAccess {
        # odum-list-ACL-files.r
        # rule to list access on all files in a collection for designated account
        checkUserInput (*Usern, $rodsZoneClient);
        *c = "/$rodsZoneClient/home/$userNameClient";

    }
*Q = select USER_ID where USER_NAME = '*Usern';
foreach (*r1 in *Q) {*Userid = *r1.USER_ID;}
# loop over files in home collection
*Coll = '*c%';
writeLine("stdout","User *Usern has access to the following files");
*rs = select DATA_ID, DATA_NAME, COLL_NAME where COLL_NAME like '*Coll';
foreach(*r in *rs) {
*fn = *r.DATA_ID;
*Coll = *r.COLL_NAME;
*File = *r.DATA_NAME;
*Path = '*Coll/*File';
# Find ACL for the file
*Query4 = select DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = '*fn';
# Loop over access controls for each file
foreach(*Row4 in *Query4) {
*Userdid = *Row4.DATA_ACCESS_USER_ID;
if(*Userid == *Userdid) {
writeLine("stdout","*Path");
}
}
}
INPUT *Usern = "lbrieger"
OUTPUT ruleExecOut

5.1.5 Delete access to files for a specified account
The following rule sets the access for a specified account to “null” for all files within a collection. Only files that originally had access permissions set for the account are processed. The rule uses the policy function:

checkUserInput
findZoneHostName

Updates to this policy are available from
http://github.com/DICE-UNC/policy-workbook/odum-delete-access.r.

deleteAccess {
  # odum-delete-access.r
  # rule to delete access on all files in a collection for designated account
  checkUserInput (*Usern, $rodsZoneClient);
  *c = "/$rodsZoneClient/home/$userNameClient";
  *q = select USER_ID where USER_NAME = '*Usern';
  foreach(*r1 in *q) {*Userid = *r1.USER_ID;}
  # loop over files in home collection
  *Coll = '*c%';
  *rs = select DATA_ID, DATA_NAME, COLL_NAME where COLL_NAME like '*Coll';
  foreach(*r in *rs) {
    *fn = *r.DATA_ID;
    *Coll = *r.COLL_NAME;
    *File = *r.DATA_NAME;
    *Path = '*Coll/*File';
    # Find ACL for the file
    *Query4 = select DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = '*fn';
    # Loop over access controls for each file
  }
}
foreach(*Row4 in *Query4) {
    *Userdid = *Row4.DATA_ACCESS_USER_ID;
    if(*Userid == *Userdid) {
        msiSetACL("default","null",*Usern,*Path);
    }
}
INPUT *Usern = "lbrieger"
OUTPUT ruleExecOut

5.1.6 Copy files, access control lists, and AVUs to a federated data grid

One way to create an archive of a collection is to copy the files to an independent data grid, along with the access controls and descriptive metadata. This policy assumes that two data grids are federated, that the path naming for files in the second data grid is the same as the path name in the primary data grid, and that user accounts from the primary data grid have been established in the second data grid. The policy copies each file from the specified collection in the primary data grid into an equivalent directory in the second data grid, copies the access controls, and copies the metadata. If an account has not been set up in the federated data grid, the ACL is not set. Currently, the AVU copy does not work and units need to be copied.

The rule uses the policy function:
checkCollInput
checkRescInput
createLogFile
findZoneHostName
isColl

Updates to this policy are available from

CopyfileACLAVU {
    # odum-copy-ACL-AVU.r
    # rule to set ACL and AVU as copy files to an archive collection
    checkRescInput (*Res, *DestZone);
    # Loop over files in a staging area, /$rodsZoneClient/home/$userNameClient/*stage
    # Put all files into collection /$DestZone/home/$userNameClient#$rodsZoneClient/*Coll
    *Src = "/$rodsZoneClient/home/$userNameClient/*Stage";
    *Dest = "/$DestZone/home/$userNameClient" ++ "#$rodsZoneClient/" ++ *Coll;
    checkCollInput (*Src);
    checkCollInput (*Dest);
    createLogFile(*Dest,"log","Check",*Res,*LPath,*Lfile,*L_FD);

    #Get USER_ID for the input user name
    *Query = select USER_ID where USER_NAME = '$userNameClient';
    *Userid = "";
    foreach(*Row in *Query) {
        *Userid = *Row.USER_ID;
    }
    if(*Userid == "") {
        writeLine("stdout","Input user name *User is unknown");
    }
fail;
}

#========== find files to stage
*Query = select DATA_NAME, DATA_ID where COLL_NAME = '*Src';
foreach(*Row in *Query) {
  *File = *Row.DATA_NAME;
  *Sdataid = *Row.DATA_ID;
  *Src1 = *Src ++ "/" ++ *File;
  *Dest1 = *Dest ++ "/" ++ *File;
#Check whether file already exists
*Query1 = select count(DATA_ID) where COLL_NAME = '*Dest' and DATA_NAME = '*File';
foreach(*Row1 in *Query1) {*DataID = *Row1.DATA_ID;}
# Move file and set access permission
if(*DataID == "0") {
  msiDataObjCopy(*Src1,*Dest1,"destRescName=Res", *Status);
  msiSetACL("default","own",$userNameClient, *Dest1);
# copy ACLs
*Query4 = select DATA_ACCESS_TYPE, DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = *Sdataid;
#Loop over access controls for the file
foreach(*Row4 in *Query4) {
  *Userdid = *Row4.DATA_ACCESS_USER_ID;
  *Datatype = *Row4.DATA_ACCESS_TYPE;
  if(*Userid != *Userdid) {
    *Query5 = select TOKEN_NAME where TOKEN_NAMESPACE = 'access_type' and TOKEN_ID = "Datatype";
    foreach(*Row5 in *Query5) {*Access = *Row5.TOKEN_NAME;}
    *Query6 = select USER_NAME, USER_ZONE where USER_ID = *Userdid;
    foreach(*Row6 in *Query6) {
      *Usern = *Row6.USER_NAME;
      *Userz = *Row6.USER_ZONE;
    }
    msiSetACL("default","*Access","*Usern#*Userz", *Dest1);
    writeLine("stdout","*Path has access control *Access for user *Usern");
  }
# copy AVUs
*Q2 = select META_DATA_ATTR_NAME, META_DATA_ATTR_VALUE, META_DATA_ATTR_UNITS where DATA_ID = *DataID;
foreach(*R2 in *Q2) {
  *Attn = *R2.META_DATA_ATTR_NAME;
  *Attv = *R2.META_DATA_ATTR_VALUE;
  *Attu = *R2.META_DATA_ATTR_UNITS;
# This policy uses a micro-service developed by the iPlant Collaborative
  msiSetAVU("-d", *Dest1, *Attn, *Attv, *Attu);
}
}
writeLine("*Lfile", "Moved file *Src1 to *Dest1");
# Delete file from staging area
msiDataObjUnlink("objPath=*Src1++++forceFlag=", *Status);
}
}
INPUT *Stage =$"stage", *Coll=$"Archive", *DestZone=$"tempZone", *Res=$"demoResc"
5.2 Normalize data to non-proprietary formats (Policy 15)
A preservation environment must ensure that the deposited records will be viewable in the future. Viable data formats will have non-proprietary or open source applications for parsing the data formats. Examples of open source formats include text files and pdf files. The archive will typically maintain a list of allowed data formats, check each file that is archived for the data format type, and create a version of the file in a sustainable format. Archives that manage persistent objects will still preserve the original data format, enabling migration to alternate data formats in the future.

5.2.1 Detection of format type
Files that have the format type included as an extension in the file name can be automatically analyzed to set the DATA_TYPE_NAME persistent state attribute. It is then possible to query DATA_TYPE_NAME to detect whether files are present with a defined data type. This policy guesses the data type based on the file extension, and then sets the DATA_TYPE_NAME persistent state variable for each file in a collection. The rule uses the policy function:

```
rule checkCollInput

setDataType {  
  # odum-set-data-type.r
  # Input parameter is:
  # Collection name
  # Output parameters are:
  # Data type set in DATA_TYPE_NAME
  # Status
  *Coll = "/$rodsZoneClient/home/$userNameClient/*Collrel";
  checkCollInput(*Coll);
  *Query = select DATA_NAME, COLL_NAME, DATA_ID where COLL_NAME like "*Coll%";
  foreach (*Row in *Query) {
    *File = *Row.DATA_NAME;
    *Collname = *Row.COLL_NAME;
    *Pathname = "*Collname/*File";
    *Objid = *Row.DATA_ID;
    *Head = "Pathname;"
    *Type = "generic;"
    *out = errormsg(msiSplitPathByKey(*Pathname, ".", *Head, *Type), *msg);
    *out2 = errormsg(msiSetDataType(*Objid, *Pathname, *Type, *Status), *msg1);
    writeLine("stdout", "File *Pathname has data type *Type");
  }
  }
```

Updates to this policy are available from
5.2.2 **Automate format type detection**
The DATA_TYPE_NAME can be automatically set on every put of a file into the data grid. The rule uses the $objPath session variable to get the file name.

Updates to the rule are available at https://github.com/DICE-UNC/policy-workbook/blob/master/acPostProcForPut-datatype.re

```
acPostProcForPut {msiSetDataTypeFromExt;}
```

5.2.3 **Identify file format extensions in a collection**
This policy generates a list of the format extensions that are used in a collection, counts the number of files with each extension, and sums the sizes of the files with each extension. The rule uses the policy functions:

- contains
- ext

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/odum-list-extensions.r.

```
listExtensions {
  # odum-list-extensions.r
  # list the extensions used in your account
  *c = "/$rodsZoneClient/home/$userNameClient%";
  summary(*c);
  }
  summary (*c) {
    *rs = select DATA_NAME, DATA_SIZE where COLL_NAME like *c;
    *res.total = str(0);
    *total.total = str(0);
    foreach(*r in *rs) {
      *fn = *r.DATA_NAME;
      *ds = *r.DATA_SIZE;
      *ext = ext(*fn);
      *res.total = str(int(*res.total) + 1);
      *total.total = str(double(*total.total) + double(*ds));
      if (contains(*res, *ext)) {
        *res.*ext = str(int(*res.*ext) + 1);
        *total.*ext = str(double(*res.*ext) + double(*ds));
      } else {
        *res.*ext = str(1);
        *total.*ext = *ds;
      }
    }
    writeLine("stdout", "ext\t\tnumber\t	avg	size\ttotal	size");
    foreach(*ext in *res) {
      if(*ext != "total") {
        *c4 = *total.*ext;
        *c1 = *ext ++ "\t";
        if (strlen("*ext") < 8) {*c1 = *c1 ++ "\t";}
        *c2 = *res.*ext ++ "\t";
        if (strlen("*res.*ext") < 8) {*c2 = *c2 ++ "\t";}
        *tot = 0;
```

68
5.3 Creation of PREMIS event data (Policy 16)

The PREMIS schema identifies events that are applied to records in an archive. The types of events include modifications to the record, usage of the record, and actions taken by the archive administrator. The pluggable architecture of iRODS version 4.1 allows each operation to be annotated with pre- and post- policy enforcement points. Information about the execution of the operation can be trapped and written to a log file. The log file can be processed to add PREMIS-style event metadata to each record. A scalable approach uses an external index to manage the PREMIS event metadata.

PREMIS metadata includes information about:

1. Data record composition, location, creating application, creation date, dependencies, format, type, size, software dependencies
2. Environment, hardware, storage medium
3. Links to permission statements, intellectual entities
4. Messages
5. Related objects, relationship type
6. Signatures, signers
7. Event types, values, sequence

The events that occur within the data management environment can be mapped to PREMIS event information:

- `relatedEventIdentifierType`
- `relatedEventIdentifierValue`
- `relatedEventSequence`

This information can be kept in an external indexing system to enable analysis, identification of the types of events that occur within the data management system,
and timelines of the events applied to a specific data record. Communication with the external indexing system is done through a message queue.

5.3.1 Creating PREMIS event information

The following rules are based on the Databook system for tracking event information about usage, data sets, and users. The rule creates a JSON document representing an access event encoded as PREMIS metadata and sends it via the Advanced Message Queue Protocol to an external indexing system.

The PREMIS event information is created using the policy functions:
- `genAccessId` which generates a URI representing this particular event.
- `jsonEncode` which encodes the data so that they can be concatenated with JSON strings.
- `sendAccess` which generates a message and sends it using AMQP
- `sendRelatedEvent` which creates a JSON document describing a related event between objects.
- `sendLinkingEvent` which creates a JSON document describing a link between two objects.

5.3.2 Sending messages over AMQP

Many indexing systems respond to messages using the Advanced Message Queue Protocol (AMQP). Policies can be used to construct AMQP messages.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dfc-amqp.re.

```python
# rules for amqp
# requires python and pika
# send a msg
amqpSend: string * string * ? -> integer
amqpSend(*Host, *Queue, *Msg) {
    writeLine("serverLog", "ampqSend: sending message ", *Msg);
    *HostArg = execCmdArg(*Host);
    *QueueArg = execCmdArg(*Queue);
    *MsgArg = execCmdArg(*Msg);
    msiExecCmd("amqpsend.py", ""*HostArg "QueueArg "MsgArg", "null", "null", "null", "Out")
}

# receive a msg
amqpRecv: string * string * output boolean * output string -> integer
    *HostArg = execCmdArg(*Host);
    *QueueArg = execCmdArg(*Queue);
    msiExecCmd("amqprecv.py", ""*HostArg "QueueArg", "null", "null", "null", "Out");
    msiGetStdoutInExecCmdOut(*Out, *Msg);
    *Emp = strlen(*Msg) == 0;
    if(!*Emp) {
        *Msg = trimr(*Msg, 
    }
}

# Xmsg to AMQP bridge
# Messages are of the format "Host:Queue:Msg", assuming that there is no ";" in Host or Queue
startXmsgAmqpBridge(*Tic, *Log) {
    delay("<EF>30s</EF>") {
        startXmsgAmqpBridge(*Tic, *Log);
    }

    amqpSend("Host", "Queue", *Msg) {
        sendAccess(*Host, *Queue, *Msg);
    }
}
```
XmsgAmqpBridge("*Tic", "*Log") {
  *Found = false;
  foreach("A in listcorerules()") {
    if("A == "amqpSend") {
      *Found = true;
      break;
    }
  }
  foreach("A in listapprules()") {
    if("A == "amqpSend") {
      *Found = true;
      break;
    }
  }
  if(!*Found) {
    msiAdmAddAppRuleStruct("amqp", "", "");
  }
  # msiXmsgServerConnect(*Conn);
  while(true) {
    if("*Log") {
      writeLine("serverLog", "waiting for message with ticket "*Tic");
    }
    if(*ErrorCode < 0) {
      if(*ErrorCode == -63000) {
        writeLine("serverLog", "no more xmessages");
        break;
      } else {
        fail(*ErrorCode);
      }
    } else {
      if("*Log") {
        writeLine("serverLog", "received xmessage "*XMsg");
      }
      *QueueMsg = triml(*XMsg, ":");
      *Host = substr(*QueueMsg, 0, strlen(*QueueMsg) - strlen(*QueueMsg) - 1);
      *Msg = triml(*QueueMsg, ":");
      *Queue = substr(*QueueMsg, 0, strlen(*QueueMsg) - strlen(*Msg) - 1);
      if("*Log") {
        writeLine("serverLog", "sending amqp message "*Host:"*Queue:"*Msg");
      }
      amqpSend("*Host", "*Queue", "*Msg");
    }
  }  
  # msiXmsgServerDisConnect(*Conn);
}@("logging", "false")

# AMQP to Xmsg bridge
# Messages are read from *Queue on *Host, and written to stream with ticket *Tic
startAmqpXmsgBridge("*Host", *Queue, "*Tic", "*Log") {
  delay("<EF>30s</EF>") {
    AmqpXmsgBridge("*Host", *Queue, "*Tic", "*Log");
  }
}

AmqpXmsgBridge("*Host", *Queue, "*Tic", "*Log") {
  *Found = false;
  foreach("A in listcorerules()") {
    if("A == "amqpRecv") {
      *Found = true;
      break;
    }
  }
  foreach("A in listapprules()") {
    if("A == "amqpRecv") {
      *Found = true;
      break;
    }
  }
}
if(!*Found) {
    msiAdmAddAppRuleStruct("amqp", "", "");
}

while(true) {
    if(*Log) {
        writeln("serverLog", "waiting for message from *Host:*Queue");
    }
    if(*ErrorCode < 0) {
        writeln("serverLog", "AMQP receive error");
        fail(*ErrorCode);
    } else if(*Emp) {
        if(*Log) {
            writeln("serverLog", "no AMQP message");
        }
        break;
    } else {
        if(*Log) {
            writeln("serverLog", "received AMQP message *Msg");
        }
        if(*Log) {
            writeln("serverLog", "sending Xmessage *Msg with ticket *Tic");
        }
        writeXMsg(str(*Tic), *Queue, *Msg);
    }
}
# msiXmsgServerDisConnect(*Conn);
}@("logging", "false")

# Xmsg to AMQP bridge which sends all Xmsgs from a channel to a queue
startXmsgAmqpBridgeOneQueue(*Tic, *Host, *Queue, *Log) {
    delay("<EF>30s</EF>") {
        XmsgAmqpBridgeOneQueue(*Tic, *Host, *Queue, *Log);
    }
}

XmsgAmqpBridgeOneQueue(*Tic, *Host, *Queue, *Log) {
    *Found = false;
    foreach("A in listcorerules()") {
        if("A == "amqpSend") {
            *Found = true;
            break;
        }
    }
    foreach("A in listapprules()") {
        if("A == "amqpSend") {
            *Found = true;
            break;
        }
    }
    if(!*Found) {
        msiAdmAddAppRuleStruct("amqp", "", "");
    }
    # msiXmsgServerConnect(*Conn);
    while(true) {
        if(*Log) {
            writeln("serverLog", "waiting for message with ticket *Tic");
        }
        if(*ErrorCode < 0) {
            if(*ErrorCode == -63000) {
                writeln("serverLog", "no more xmessages");
            } else {
...
fail(*ErrorCode);
}
#else {
  if(*Log) {
    writeLine("serverLog", "received xmessage "XMsg");
  }
  amqpSend(*Host, *Queue, "XMsg");
}
#endif

5.4 Automation of user submission agreements (Policy 17)
When files are loaded into a staging area, processing steps can be applied before the
file is moved to the archival location. An example is the acquisition of a signed user
submission agreement. A user submission agreement typically specifies that the
user owns the copyright to the file, has the authority to submit the file to an archive,
and agrees to a set of access permissions for the file. This can be automated through
use of E-mail, web forms, or formal hard copy submission agreements.

5.4.1 Staging of files with a user submission agreement
Files can be moved from a staging area into an archive when the presence of a user
submission agreement is checked. This policy assumes that a separate collection is
formed within the staging area, and that the user submission agreement has been
associated as an attribute on the collection name. As in the previous policy, the
variable name “Use_Agreement” is checked to see if the value is “RECEIVED”. In this
case, the collection name is checked instead of the USER_NAME. The rule uses the
policy function:
  checkCollInput

Updates to this policy are available from

myStagingRule {
  # odum-stage-ag.r
  # Loop over files in a staging area, /$rodsZoneClient/home/$userNameClient/*stage
  # Check that the variable “Use_Agreement” on the collection has the value “RECEIVED”
  # If true, put all files into collection /$rodsZoneClient/home/$userNameClient/*Coll
  *Src = "/$rodsZoneClient/home/$userNameClient/" ++ *Stage;
  *Dest = "/$rodsZoneClient/home/$userNameClient/" ++ *Coll;
  checkCollInput (*Src);
  checkCollInput (*Dest);
  *Query1 = select META_COLL_ATTR_VALUE where COLL_NAME = 'Src' and
             META_COLL_ATTR_NAME = 'Use_Agreement';
  foreach (*Row1 in *Query1) {
    *Use = *Row1.META_COLL_ATTR_VALUE;
  }
  if (*Use == "RECEIVED") {
    *Query2 = select DATA_NAME where COLL_NAME = 'Src';
    foreach(*Row2 in *Query2) {
      *File = *Row2.DATA_NAME;
      *Src1 = "Src " ++ *File;
  }
*Dest1 = *Dest ++ "/" ++ *File;
#Check whether file already exists
*Q3 = select count(DATA_NAME) where COLL_NAME = '*Dest' and DATA_NAME = '*File';
foreach (*R3 in *Q3) { *DataID = *R3.DATA_NAME; }
# Move file and set access permission
if(*DataID == "0") {
  msiDataObjRename (*Src1, *Dest1,"0", *Status);
  if (*Status == "0") {
    writeLine("stdout", "Moved file *Src1 to *Dest1");
  }
  else {
    writeLine("stdout", "File *Src1 was not moved");
  }
}
}
INPUT *Stage ="stage", *Coll="Rules"
OUTPUT ruleExecOut

5.5 Automatic Checksums (Policy 18)
The BagIt technology encapsulates data in a container before transport over the network. Within the container, a manifest file is added that provides a checksum for each enclosed file. The checksum can be extracted, compared to a new checksum generated upon receiving the file, and verified to ensure that the data were not corrupted on transport. The checksum can be recorded as a metadata attribute on the file, DATA_CHECKSUM, and used in the future to verify file integrity.

5.5.1 Creating a BagIt file
This rule generates a bag (tar file) containing a manifest, a list of checksums, and the files contained within a specified collection.

The generateBagIt rule creates the equivalent of a Submission Information Package. Extensions would be the inclusion of descriptive metadata, provenance metadata, and structural metadata. The rule uses the policy function:

    checkCollInput

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/odum-bagit.r.

generateBagIt { 
# ---------------------------------------------
# odum-bagit.r
# ---------------------------------------------

# Terrell Russell
# University of North Carolina at Chapel Hill
# - August 2010
# - Requires iRODS 2.4.1 or greater
# - Conforms to BagIt Spec v0.96
#
# ---------------------------------------------
# - creates NEWBAGITROOT
### - writes bagit.txt to NEWBAGITROOT/bagit.txt
### - rsyncs existing BAGITDATA to NEWBAGITROOT/data
### - generates payload manifest file of NEWBAGITROOT/data
### - writes payload manifest to NEWBAGITROOT/manifest-md5.txt
### - writes tagmanifest file to NEWBAGITROOT/tagmanifest-md5.txt
### - creates tarfile of new bag for faster download
### - gets filesize of new tarfile
### - outputs report and suggested download procedures
### - writes to rodsLog

# -- --------------------------------------------------------

checkCollInput (*BAGITDATA);
### - creates NEWBAGITROOT
msiCollCreate (*NEWBAGITROOT,"1",*Status);
msiStrlen (*NEWBAGITROOT,*ROOTLENGTH);
OFFSET = int (*ROOTLENGTH) + 1;
### - writes bagit.txt to NEWBAGITROOT/bagit.txt
writeLine ("stdout","BagIt-Version: 0.96");
writeLine ("stdout","Tag-File-Character-Encoding: UTF-8");
msiDataObjCreate("*NEWBAGITROOT" ++ "/bagit.txt",null,*FD);
msiDataObjWrite(*FD,"stdout",*WLEN);
msiDataObjClose (*FD,*Status);
msiFreeBuffer("stdout");
### - rsyncs existing BAGITDATA to NEWBAGITROOT/data
msiCollRsync (*BAGITDATA,"*NEWBAGITROOT" ++ "/data",null,"IRODS_TO_IRODS",*Status);
### - generates payload manifest file of NEWBAGITROOT/data
*NEWBAGITDATA = *
ContInxOld = 1;
*Condition = "COLL_NAME like "*NEWBAGITDATA%%";
msiMakeGenQuery("DATA_ID, DATA_NAME, COLL_NAME",*Condition,*GenQInp);
msiExecGenQuery (*GenQInp,*GenQOut);
msiGetContInxFromGenQueryOut (*GenQOut,*ContInxNew);
while (*ContInxOld > 0) {
foreach (*GenQOut) {
 msiGetValByKey (*GenQOut,"DATA_NAME",*Object);
 msiGetValByKey (*GenQOut,"COLL_NAME",*Coll);
 *FULLPATH = "*Coll" ++ "/" ++ "*Object";
 msiDataObjChksum(*FULLPATH,"forceChksum=",*CHKSUM);
 msiSubstr (*FULLPATH,str(*OFFSET),null,*RELATIVEPATH);
 writeString ("stdout",*RELATIVEPATH);
 writeLine ("stdout","*CHKSUM")
}
*ContInxOld = *ContInxNew;
if (*ContInxOld > 0) {msiGetMoreRows (*GenQInp,*GenQOut,*ContInxNew);} }
### - writes payload manifest to NEWBAGITROOT/manifest-md5.txt
msiDataObjCreate("*NEWBAGITROOT" ++ "/manifest-md5.txt",null,*FD);
msiDataObjWrite(*FD,"stdout",*WLEN);
msiDataObjClose (*FD,*Status);
msiFreeBuffer("stdout");
### - writes tagmanifest file to NEWBAGITROOT/tagmanifest-md5.txt
writeString("stdout","bagit.txt ");
msiDataObjChksum("NEWBAGITROOT" ++ "/bagit.txt", "forceChksum", "CHKSUM");
writeLine("stdout","CHKSUM");
writeString("stdout","manifest-md5.txt ");
msiDataObjChksum("NEWBAGITROOT" ++ "/manifest-md5.txt", "forceChksum", "CHKSUM");
writeLine("stdout","CHKSUM");
writeString("stdout","manifest-md5.txt ");
msiDataObjChksum("NEWBAGITROOT" ++ "/manifest-md5.txt","forceChksum", "CHKSUM");
writeLine("stdout","CHKSUM");
msiDataObjCreate("NEWBAGITROOT" ++ "/tagmanifest-md5.txt", "null", "FD");
msiDataObjWrite("FD","stdout","WLEN");
msiDataObjClose("FD","Status");
msiFreeBuffer("stdout");

### - creates tarfile of new bag for faster download
msiTarFileCreate("NEWBAGITROOT" ++ ".tar", "null", "Status");

### - gets file size of new tarfile
msiSplitPath("NEWBAGITROOT" ++ ".tar", "null", "TARFILENAME");
msiMakeQuery("DATA_SIZE", "COLL_NAME like "*Coll%%' AND DATA_NAME = *TARFILENAME", "Query");
msiExecStrCondQuery("Query", "E");
foreach(*E)
{
  msiGetValByKey(*E,"DATA_SIZE","FILESIZE");
  *Isize = int("FILESIZE");
  if(*Isize > 1048576) {
    *PRINTSIZE = *Isize / 1048576;
    *PRINTUNIT = "MB";
  } else {
    if(*Isize > 1024) {
      *PRINTSIZE = *Isize / 1024;
      *PRINTUNIT = "KB";
    } else {
      *PRINTSIZE = *Isize;
      *PRINTUNIT = "B";
    }
  }
}

### - outputs report and suggested download procedures
writeLine("stdout","");
writeLine("stdout","Your BagIt bag has been created and tarred on the iRODS server:");
writeLine("stdout","NEWBAGITROOT.tar - "PRINTSIZE *PRINTUNIT");
writeLine("stdout","");
msiSplitPath("NEWBAGITROOT" ++ ".tar","null", "TARFILENAME");
writeLine("stdout","To copy it to your local computer, use:");
writeLine("stdout","iget -Pf "NEWBAGITROOT.tar *TARFILENAME");
writeLine("stdout","");
msiWriteRodsLog("BagIt bag created: NEWBAGITROOT <~ BAGITDATA", "Status");

INPUT "BAGITDATA=$"/rodsZoneClient/home/$userNameClient/sub1",
"NEWBAGITROOT=$"/rodsZoneClient/home/$userNameClient/bagit"
OUTPUT ruleExecOut

5.6 Automated capture of Provenance/contextual metadata (Policy 19)
Provenance and contextual metadata can be associated with files as metadata attributes. The source of the metadata may be an XML file, or a text file, or a structure within each data file. An automated process to acquire the metadata
would parse the metadata source file, and load the metadata as attributes on each archived file. Examples of this approach are provided in Chapter 4.6.

5.6.1 Provenance for administrative policies

Provenance can also be tracked for execution of administrative policies. Workflow structured objects implement automated capture of provenance information for each execution of a workflow. The workflow file is of data type 'msso' and uses the dot-extension '.mso'. The workflow file is registered into iRODS and can be shared, executed, and re-executed. The workflow language is the same as that of the '.r' file used by irule command, but need not have the INPUT and OUTPUT statements. Policies can be stored as workflows, with each execution of the workflow tracked by the data grid.

For each workflow file, one associates a structured object that implements an iRODS collection-type environment for tracking executions of the workflow. All files associated with a workflow execution are stored under this structured object called the Workflow Structured Object (WSO). One can view the WSO akin to an iRODS collection with a hierarchical structure. At the top level of this structures, one stores all the parameter files needed to run the workflow, as well as any input files and manifest files that are needed for the workflow execution. Beneath this level, a set of run directories is created which actually house the results of an execution. Hence, one can view the WSO as a complete structure that captures all aspects of a workflow execution. In iRODS the WSO is created as a mount point in the iRODS logical collection hierarchy. This is similar to a mounted collection but of type "msso". One uses the imcoll command to create this mount point. We use WSO and MSSO (micro-service structured object) synonymously for historic reasons since the need and idea for WSO/MSSO came from the usage experience for Micro-Service Objects (MSO).

Apart from the workflow file there is one other important file called the parameter file (with dot-extension '.mpf') which contains information needed for executing the workflow. We separated the parameter file from the workflow file such that one can associate multiple parameter files with a workflow and use them for executing with different input values. The parameter files contains values for workflow *variables that are used in the workflow execution. It also contains information about files that need to staged in before the execution and staged out for archiving after the execution. It also contains directives for the workflow execution engine. The parameter files as well as any input files can be ingested into the WSO using normal icommands such as iput.

When a parameter file is ingested into a WSO, a run file is automatically created which can be used to run the parameter file with the associated workflow. When a workflow execution occurs a run directory is created for storing the results of this run. Depending upon the directives in the parameter file, older results are versioned out or discarded after a successful workflow execution. These version directories can be listed and accessed using the normal icommands such as ils and iget.
Workflows can be called from within other workflows. This feature allows one to chain workflows. This can be done in two ways. One is by opening another workflow parameter file inside a workflow and using the data returned from this as normally done for accessing files in iRODS. A second way of running a workflow inside another is to call it through a special policy called "acRunWorkFlow". The first way is useful if the output file from a workflow is very large and needs to process multiple buffer read calls. The second way is useful when the returned data is less than 32 MB in size. Samples of both versions are shown below.

Updates to this workflow are available from http://github.com/DICE-UNC/policy-workbook/odum-eCWkflow.mss.

Sample Workflow file: **eCWkflow.mss**

```text
#Input parameters:
# Name of *File1 - first output file written by the workflow
# Name of *File2 - second output file written by the workflow
#Output parameter is:
# None
#Output from running the example is:
# message about completion written to stdout
# This workflow executes the file called myWorkFlow twice with two different input values
# This is an executable file that is located in bin/cmd directory of the iRODS server.
# It creates an output file using the value given in the second argument.
# The workflow also prints to stdout the statement about when the execution occurred.
testWorkflow {
  odum-eCWkflow.mss
  msiExecCmd("myWorkFlow", *File1, "null", "null", "null", *Result1);
  msiExecCmd("myWorkFlow", *File2, "null", "null", "null", *Result2);
  msiGetFormattedSystemTime(*myTime, "human", "%d-%d-%d %ldh:%ldm:%lds");
  writeLine("stdout", "Workflow Executed Successfully at *myTime");
}
```

Sample Parameter file used with eCWkflow.ms: **eCWkflow.mpf**

```text
#Comments
#
#FileName should be StarVariableName occurring
# either in INPUT of the msso file or in INPARAM of this file.
#Please identify all file names as they will be helpful for later metadata extraction
#FILEPARAM fileStarVariableName
#DIRPARAM collStarVariableName
#
#INPARAM paramName=paramValue
#INPARAMINFO paramName, paramType=type, paramUnit=unit, valueSize=size,
Comments=comments
# parameters used by the workflow
# In this case There are two files and another string value parameter.
INPARAM *File1="OutFile3"
INPARAM *File2="OutFile4"
INPARAM *Aval="test"
```
# Identify files that are used in input params - needed to stage back outputs.
FILEPARAM *File1
FILEPARAM *File2
#
# Identify the stage area where the workflow execution is performed
# by default it is performed at the "bin" directory of the iRODS server.
# This is needed if one is using msiExecCmd micro-service as part of the workflow.
#STAGEAREA bin
#
# Stage in files from anywhere in iRODS to the "stage area"
# myData is a file located in the WSO and photo.JPG is a file somewhere else in iRODS.
STAGEIN myData
STAGEIN /raja8/home/rods/photo.JPG
#
# Stage back additional files created as part of run
# COPYOUT - will leave a copy in the "stage area" and make a copy in iRODS WSO
# - helpful if it is needed by subsequent workflow execution
# STAGEOUT - will move file from "stage area" to iRODS WSO
# In this case we are archiving the two files myData and photo.JPG as well as the
# "myWorkFlow" file used by the workflow execution.
COPYOUT myWorkFlow
STAGEOUT myData
STAGEOUT photo.JPG
#
#The next set of statements provide directives to the workflow system.
# CHECKFORCHANGE is used for testing where the file being checked has changed since
# the previous execution of the workflow. If the file is modified/touched then the workflow
# is executed. If none of the files are changed, then the workflow is not executed. If
# directed, the file from previous execution is "sent back" to the client.
# NOVERSION is used when versioning of old results is not needed.
# CLEANOUT is used to clear the stage area after execution.
#
CHECKFORCHANGE /raja8/home/rods/photo.JPG
CHECKFORCHANGE myData

Just for full information disclosure the executable for **myWorkFlow** is also provided below.

```
#!/bin/sh
# Just a test to copy an existing file
# one may look at this as taking a file and creating a new one possibly after conversion
# mycp is a file that takes tt as input and creates a new output file
cmd/mycp cmd/tt "$1"
```

Calling a workflow from another workflow is possible. The following example shows a workflow call embedded as an object open in the sample workflow shown above.

Updates to this workflow are available from

testWorkflowCall1 {
# odum-testWorkflowCall1.mss
    msiExecCmd("myWorkFlow", *File1, "null", "null", "null", *Result1);
The next example shows the same action using a rule and is useful when reading small files. Updates to this workflow are available from http://github.com/DICE-UNC/policy-workbook/odum-testWorkflowCall2.mss.

testWorkflowCall2 {
    # odum-testWorkflowCall2.mss
    msiExecCmd("myWorkFlow", *File1, "null", "null", "null", *Result1);
    msiExecCmd("myWorkFlow", *File2, "null", "null", "null", *Result2);
    acRunWorkFlow("/raja8/home/rods/msso/mssop1/mssop1.run++++openFiles=O_RDONLY", *S_FD);
    msiDataObjOpen("objPath=/raja8/home/rods/msso/mssop1/mssop1.run++++openFiles=O_RDONLY", *S_FD);
    msiDataObjRead(*S_FD,*Len,*R_BUF);
    msiDataObjClose(*S_FD,*Status2);
    # Process *R_BUF contents
    writeLine("stdout", "Workflow Executed Successfully at *myTime");
}

The steps for using a workflow object are outlined below.
First create a new collection and ingest the workflow file

```
imkdir /dfctest/home/rodsAdmin/workflow
iput -D "msso file" ./dfcDemoWkFlow.mss
```

Create a new collection and mount that collection as a Workflow Structured Object associated with the workflow file. The collection that is mounted as an MSO for a workflow can be anywhere in iRODS. As can be seen, one can have more than one such structure mounted for a workflow file. The name of the collection need not be related to the name of the workflow file.

```
imkdir /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow
imcoll -m msso /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow.mss
```

Create a new collection and mount that collection as a Workflow Structured Object associated with the workflow file. The collection that is mounted as an MSO for a workflow can be anywhere in iRODS. As can be seen, one can have more than one such structure mounted for a workflow file. The name of the collection need not be related to the name of the workflow file.

```
imkdir /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow
imcoll -m msso /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow.mss
```

Ingest a parameter file in the WSO collection. One can ingest more than one parameter file also in the same WSO collection. A run file for each parametric file is automatically created.

```
iput dfcDemoWkFlow.mpf /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow
iput dfcDemoWkFlow2.mpf /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow
```

One can ingest other files (such as input files) that are needed for workflow execution.

```
iput myData /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow/myData
```

One can perform ils on the WSO collection. It will show the two parameter files as well as run files that are automatically created for each of them. Note that the name of the run file is based on the file name of the parametric file.

```
is /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow
```

One can perform ils on the WSO collection. It will show the two parameter files as well as run files that are automatically created for each of them. Note that the name of the run file is based on the file name of the parametric file.

```
is /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow
```
One can perform other commands also on the WSO collection. The iget command will show the contents of the file.

```
  icd /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow
  ils -l
  iget ../dfcDemoWkFlow.mss -
  iget dfcDemoWkFlow.mpf -
  iget dfcDemoWkFlow2.mpf -
  iget myData -
```

To execute the workflow using a parametric file, perform an access on the associated run file. Instead of showing what is in the "run" file, this iget action executes the workflow using the associated parametric file and stores the results. The iget returns a file back to the client. By default the stdout from execution of the workflow is returned. If one needs a different file to be returned, one can set that up as part of the workflow file or the parametric file using the directive "SHOW".

```
  iget dfcDemoWkFlow.run -
  Workflow Executed Successfully at 2012-9-20 11h:28m
```

The execution of the workflow also creates a new directory as part of the WSO structure and stores the results of the execution (as per the directives in the .mpf parametric file). This can be seen by performing a listing of the directory which will be named after the parametric file.

```
  ils
  /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow:
  dfcDemoWkFlow.run
  dfcDemoWkFlow.mpf
  dfcDemoWkFlow2.run
  dfcDemoWkFlow2.mpf
  myData
  C:/dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow/dfcDemoWkFlow.runDir
```

Listing the runDir will show the results of the run. Compare this with the directive in the parametric file above.

```
  ils -l dfcDemoWkFlow.runDir
  /dfctest/home/rodsAdmin/workflow/dfcDemoWkFlow/dfcDemoWkFlow.runDir:
  rodsAdmin  mssoSt demoResc  11 2012-09-20.11:28 & myData
  rodsAdmin  mssoSt demoResc  99 2012-09-20.11:28 & myWorkflow
  rodsAdmin  mssoSt demoResc  20 2012-09-20.11:28 & OutFile1
  rodsAdmin  mssoSt demoResc  20 2012-09-20.11:28 & OutFile2
  rodsAdmin  mssoSt demoResc 1181588 2012-09-20.11:28 & photo.JPG
  rodsAdmin  mssoSt demoResc  52 2012-09-20.11:28 & stdout
```

Any of the files in the runDir directory can be accessed using the iget command. Also, one can have whole directories stored under the runDir. If you run the workflow again without changing the input, the workflow is not actually executed. Instead the contents of the old stdout is sent back to the client. Also there will be no new files created.

```
  iget dfcDemoWkFlow.run -
  Workflow Executed Successfully at 2012-9-20 11h:30m
```

This is because neither the input files nor the workflow system have changed and as per directive, it will not re-execute the workflow. If we overwrite one of the input files, the workflow will be executed. Since the NOVERSION directive is not in the
parameter file, the older files will be versioned and the new files created in the runDir directory.

```
iget dfcDemoWkFlow.run -
Workflow Executed Successfully at 2012-9-20 11h:30m
ils -l dfcDemoWkFlow.runDir
```

As can be seen below, the older execution files are stored under dfcDemoWkFlow.runDir0

```
ils -l dfcDemoWkFlow.runDir0
```

One can run the workflow with another parametric file and it will be placed in a new directory.

```
iget dfcDemoWkFlow2.run -
Workflow Executed Successfully at 2012-9-20 11h:31m
```

```
Note that the name of the output files are different in the second run as the names were changed in dfcDemoWkFlow2.mpf

5.7 Federation – periodically copy data (Policy 20)
A policy for copying data between two federated data grids was provided in section 4.7.3. The policy can be turned into a periodically executed rule by adding a delay command that executes the policy every week.

This rule takes all files in a “stage” directory on the first data grid, copies them to an “Archive” directory on the second data grid, and deletes the file from the first data grid. The rule also logs all of the actions and writes the log to a directory in the second data grid. The rule uses the policy functions:

```
checkCollInput
checkRescInput
createLogFile
findZoneHostName
isColl
```

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/odum-periodic-stage.r.

```plaintext
myStagingRule {
    # odum-periodic-stage.r
    # Loop over files in a collection, *Src
    # Put all files into a staging collection. *Dest
    checkCollInput (*Src);
    checkCollInput (*Dest);
    checkRescInput (*Res, *DestZone);
    delay ('<PLUSET>1m</PLUSET><EF>7d</EF>') {
        stage(*Src, *Res, *Dest);
    }
    stage(*Src, *Res, *Dest) {
        *Len = strlen(*Src);
        #=============get current time, Timestamp is YYYY-MM-DD.hh:mm:ss ==============
        msiGetSystemTime("TimeA","unix");
        #=============== create a collection for log files if it does not exist ================
        createLogFile("*Src","log","Check",*Res,LPath,Lfile,L_FD);
        #========= find files to stage
        *Query = select DATA_NAME, DATA_CHECKSUM,COLL_NAME where COLL_NAME like '*Src%';
        foreach(*Row in *Query) {
            *File = *Row.DATA_NAME;
            *Check = *Row.DATA_CHECKSUM;
            *Coll = *Row.COLL_NAME;
            if (*Coll != LPath ) {
                *L1 = strlen(*Coll);
                *Src1 = *Coll ++ "/" ++ *File;
                *C1 = substr(*Coll,*Len,*L1);
            }
```
\begin{verbatim}
if(strlen(*C1)==0) {
    *DestColl = *Dest;
    *Dest1 = *Dest ++ "/" ++ *File;
} else {
    *DestColl = *Dest ++ *C1;
    *Dest1 = *Dest ++ *C1 ++ "/" ++ *File;
}
isColl(*DestColl, *Lfile, *Status);
if(*Status >= 0)
{
    msiDataObjCopy(*Src1,*Dest1,"destRescName=*Res++++forceFlag=",*Status);
    msiSetACL("default","own","rwmoore#testZone",*Dest1);
    msiDataObjChksum(*Dest1, "forceChksum=",*Chksum);
    if(*Check != *Chksum) {
        writeLine("*Lfile", "Bad checksum for file *Dest1");
    } else {
        writeLine("*Lfile", "Moved file *Src1 to *Dest1");
        writeLine("*Lfile", "Moved file *Src1 to *Dest1");
    }
}
}
\end{verbatim}

INPUT *Res=$"demoResc", *DestZone =$"tempZone",
*Src=$"/rodsZoneClient/home/$userNameClient/stage", *Dest =$"/
*DestZone/home/$userNameClient#$rodsZoneClient/stage"
OUTPUT ruleExecOut

5.8 De-identification of Data (Policy 25)
This is crucial for all repositories in all fields when human subjects data are
involved. Information related to addresses, social security numbers, and credit cards
has to be identified and removed. The identification of personally identified data
within submitted digital objects may be part of a user submission agreement. The
ability to automate the detection is essential when researchers submit material.

5.8.1 BitCurator based processing
The BitCurator project brings in a series of open source digital forensics tools and
techniques to collecting institutions, to preserve their born-digital collections [6].
iRODS (Integrated rule-oriented data system) is a data-grid software system, where
users can build sharable collections from data distributed across file systems and
tape archives[9]. This project integrates the two technologies, allowing a user of
iRODS to run the BitCurator tools in an iRODS environment and copy the resulting
reports into the iRODS grid. This document lists the BitCurator tools that are
integrated into iRODS and a overview of each tool along with a description on how
to use it. The tools are run on an iRODS server, requiring an installation by the data
grid administrator.

The prerequisite for running the Bitcurator tools on a media or any set of files is to
use the tool “Guymager” (http://guymager.sourceforge.net/) and generate an image
in the .aff or .E01 format.
5.8.1.1 Generate Digital Forensics XML file

This utility uses the BitCurator Fiwalk tool, takes an image in the .aff or E01 form and generates an XML file. As per [7], “Digital Forensics XML (or DFXML) is a metadata schema designed to facilitate the sharing of structured information produced by forensic tools. DFXML is an attempt to standardize abstractions by providing a formalized language for describing forensic processes”. Refer to [7] for more details.

The command to be executed is located in the directory irods/server/bin/cmd/fiwalk.

This rule Invokes the Fiwalk tool to generate the XML output of the given disk image.

Input Parameter is: Image File path
Output Parameter is: XML File path

**Command Structure:**
```
irule -F rulesiBcGenerateXml.r "*outXmlFile='/Path/to/xmlfile'
"*image='/path/to/image.aff'
```

**Rule Location:**
odum-bcGenerateFiwalkRule.r

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/odum-bcGenerateFiwalkRule.r.

```bash
bcGenerateFiwalkRule { 
  # odum-bcGenerateFiwalkRule.r
  # Command to be executed is located in directory irods/server/bin/cmd/fiwalk
  # This rule Invokes the Fiwalk tool to generate the XML output of the given
  # disk image.
  #
  # # Input Parameter is:
  # #  Image File path
  # Output Parameter is:
  # #  XML File path
  #
  # # Example:
  # # irule -F rulesiBcGenerateXml.r **outXmlFile='/Path/to/xmlfile'** **image='/path/to/image.aff'
  #  *Cmd="Fiwalk";
  #  *timeStamp = double (time());
  #  msiSplitPath(*image, *Coll, *File);
  #  # Query the metadata catalog check whether "image" exists
  #  *Q1 = select DATA_NAME) where DATA_NAME = **File** and COLL_NAME = **Coll**;
  #  foreach(*R1 in *Q1) {
  #    *Num = *R1.DATA_NAME;
  #    if(*Num == "0") {
  #      writeLine("stdout", "Please enter a filename");
  #      fail;
  #    }
  #  }
  #
  #  # Image exists
  #  # Now Make a query to get the absolute path to the image and the resource name
  #  *Query = select DATA_PATH, DATA_RESC_NAME where DATA_NAME = **File** and COLL_NAME = **Coll**;
  #  foreach (*row in *Query) {
  #    *Path = *row.DATA_PATH;
  #    *Resource = *row.DATA_RESC_NAME;
  #    writeLine("stdout", "Path = **Path, Resource= **Resource");
  #  }
```
# Make another query for IP Address of the resource
# RESC_LOC: Resource IP Address
# DATA_RESC_NAME: Logical name of storage resource
*Query2 = select RESC_LOC where DATA_RESC_NAME = "Resource";
foreach (*row in *Query2) {
    *Addr = *row.RESC_LOC;
    writeLine("stdout", "ADDR = *Addr, Resource = *Resource");
}

*Arg1 = execCmdArg("-f");
*Arg2 = execCmdArg("-X");
*prefixStr = "*timeStamp$userNameClient";
*tempStr = "/tmp/*prefixStr " + "outXmlFile";

# Shellscript will remove /tmp/*timeStamp$userNameClient*

*Arg3 = execCmdArg(*tempStr);
*Arg4 = execCmdArg(*Path);
writeLine("stdout", "Running Fiwalk Command...");
writeLine("stdout", "Command: *Cmd *Arg1 *Arg2 *Arg3 *Arg4");

if (errorcode(msiExecCmd(*Cmd,"*Arg1 *Arg2 *Arg3 *Arg4", "null", "null", "Result") < 0) {
    if(errormsg(*Result,*msg)==0) {
        msiGetStderrInExecCmdOut(*Result,*Out);
        writeLine("stdout", "ERROR: *Out");
    } else {
        writeLine("stdout", "Result msg is empty");
    }
} else {
    # Command executed successfully
    msiGetStdoutInExecCmdOut(*Result,*Out);
    writeLine("stdout", "Output is *Out");

    # Clean up the temporary files
    cleanup(*Addr,*tempStr,*outXmlFile,*prefixStr,*status);
}

# Function: cleanup: Calls a script to remove the temporary files created
# in /tmp
cleanup: input string * input string * input string * input string * output integer -> integer
cleanup(*Addr,*tempStr,*outXmlFile,*prefixStr,*status) {
    remote("Addr","null") {
        *local = "localPath=*tempStr++++forceFlag="; #str(*options);
        writeLine("stdout","cleanup: local: " + *local);
        writeLine("stdout","cleanup: outXmlFile: " + *outXmlFile);
        writeLine("stdout","cleanup: tempStr: " + *tempStr);

        msiDataObjPut(*outXmlFile,"null", *local,*status);
        *Arg1 = execCmdArg(*prefixStr);
        msiExecCmd("tmpCleanup",*Arg1,"null", "null", "null", *Result);
    }
}

INPUT *outXmlFile="/AstroZone/home/pixel/bcfiles/xmlfile",
"image="/AstroZone/home/pixel/bcfiles/charlie-work-usb-2009-12-11.afl"
OUTPUT ruleExecOut

Command examples:
1. irule -F rulemsiBcGenerateXml.r

Default parameters can be modified by changing the following line
with appropriate values:
INPUT *outXmlFile="/AstroZone/home/pixel/bcfiles/xmlfile",
2. `irule -F rulemsiBcGenerateXml.r "*outXmlFile='/home/xmlfile'" 
"*image='/home/test.aff'"

**Files:**

- **Local File System:**
  The following file resides on the Local File System:
  
  `$iRODS/server/bin/cmd/fiwalk`

- **iRODS Grid:**
  Executing this rule creates the following file on the grid:
  
  `$iRODS_grid/<xmlfile>`

**Implementation notes:**

The `fiwalk` tool, an executable file, is copied to `iRODS/server/bin/cmd` directory:

```
  cp /usr/local/bin/fiwalk iRODS/server/bin/cmd/fiwalk
```

### 5.8.1.2 Bulk Extractor

The “`bulk_extractor` is a computer forensics tool that scans a disk image, a file, or a directory of files and extracts useful information without parsing the file system or file system structures. The results can be easily inspected, parsed, or processed with automated tools.” [8] This tool takes the disk image (the `.aff` file) as an input and generates an output directory in the specified location, containing a text file for each of the features located in the input image. For more information on Bulk Extractor scanners, refer to the following URLs:

- [http://www.forensicswiki.org/wiki/Bulk_extractor](http://www.forensicswiki.org/wiki/Bulk_extractor)

The command to be executed is located in directory `irods/server/bin/cmd/bulk_extractor`

The execution command is

```
  bulk_extractor <image.aff> -o <output directory>
```

Input Parameter is: Image File path
Output Parameter is: File Path for Feature Files

**Command Structure:**

```
  irule -F rulemsiBcExtractFeatureFiles "*image='/path/to/image.aff'"
  "outFeatDir='/path/to/outdir'"
```

Updates to this policy are available from


```
bExtractFeatureFilesRule { 
  # odum-bcExtractFeatureFilesRule.r
  # Command to be executed located in directory irods/server/bin/cmd/bulk_extractor
  # This rule reads the disk image and generates a number of feature files in
  # the specified output directory. It invokes the Bulk Extractor tool.
  # Input Parameter is:
  #   Image File path
  # Output Parameter is:
```
File Path for Feature Files
#
# Example:
# bulk_extractor
# ~/Research/TestData/M57-Scenario/usbflashdrives/jo-work-usb-2009-12-11.aff
# -o ~/Research/TestData/BEOutputs/jow-output
# Rule:
# irule -F rulemsiBcExtractFeatureFiles "*image=’/path/to/image.aff’"
"outFeatDir=’/path/to/outdir’"

*Cmd="bulk_extractor";
*timeStamp = double (time());
msiSplitPath(*image, *Coll, *File);  
# Query the metadata catalog
# First check that the image exists
*Q1 = select count(DATA_ID) where DATA_NAME = ’*File’ and COLL_NAME = ’*Coll’;
foreach (*R1 in *Q1) {
    *Num = *R1.DATA_ID;
    if (*Num == ”0”)
    {
        writeLine(”stdout”, ”Please enter a filename”);
        fail;
    }
}

# Image exists
# Now Make a query to get the path to the image and the resource name
# DATA_PATH: Physical path name for digital object in resource
# DATA_RESC_NAME: Logical name of storage resource
*Query = select DATA_PATH, DATA_RESC_NAME where DATA_NAME = ’*File’ and COLL_NAME = ’*Coll’;
foreach (*row in *Query)
{
    *Path = *row.DATA_PATH;
    *Resource = *row.DATA_RESC_NAME;
    writeLine(”stdout”, ”Path = *Path, Resource= *Resource”);
}

# Make another query for IP Address of the resource
# RESC_LOC: Resource IP Address
# DATA_RESC_NAME: Logical name of storage resource
*Query2 = select RESC_LOC where DATA_RESC_NAME = ’*Resource’;
foreach (*row in *Query2) {
    *Addr = *row.RESC_LOC;
    writeLine(”stdout”, ”ADDR = *Addr, Resource= *Resource”);
}

*prefixStr = ”*timeStamp$userNameClient”;
*tempStr = ”/tmp/*prefixStr” + ”outFeatDir”;

*Arg1 = execCmdArg(*Path);  # Image
*Arg2 = execCmdArg(”-o”);  
*Arg3 = execCmdArg(*tempStr);  # Output Feature Directory

writeLine(”stdout”, ”Running Bulk Extractor Tool…”);
writeLine(”stdout”, ”Command: *Cmd *Arg1 *Arg2 *Arg3”);
if (errorcode(msiExecCmd(*Cmd,"*Arg1 *Arg2 *Arg3", "null", "image", "null",*Result)) < 0) {
    if(errormsg(*Result,msg)==0) {
        msiGetStderrInExecCmdOut(*Result,*Out);
        writeLine("stdout","ERROR:*Out");
    } else {
        writeLine("stdout","Result msg is empty");
    }
} else {
    # Command executed successfully
    msiGetStdoutInExecCmdOut(*Result,*Out);
    writeLine("stdout","Output is *Out");

    # Clean up the temporary files
    cleanup(*Addr,*tempStr,*outFeatDir,*prefixStr,*status);
}

# Function: cleanup: Calls a script to remove the temporary files created
# in /tmp
cleanup: input string * input string * input string * input string * output integer -> integer
cleanup(*Addr,*tempStr,*outFeatDir,*prefixStr,*status) {

    writeLine("stdout","Cleanup: Moving *tempStr to *outFeatDir");
    remote(*Addr,"null") {
        *local = "localPath=*tempStr++++forceFlag="; #str(*options);
        writeLine("stdout","cleanup: local: *local");
        writeLine("stdout","cleanup: outFeatDir: *outFeatDir");
        writeLine("stdout","cleanup: tempStr: *tempStr");

        msiDataObjPut(*outFeatDir,"null",*local,*status);
        *Arg1 = execCmdArg(*prefixStr);
        msiExecCmd("tmpCleanup",*Arg1,"null","null","null",*Result);
    }
}

INPUT *image="/OdumStagingZone/home/bitcurator/bitcurator_tmp/18335",
*outFeatDir="/OdumStagingZone/home/bitcurator/bitcurator_output"
OUTPUT ruleExecOut

Command examples:
1. irule -F rulesmsiBcExtractFeatureFiles.r
   Default parameters can be modified by changing the following line:
   INPUT *image="/AstroZone/home/pixel/bcfiles/charlie-work-usb-2009-12-11.aff", *outFeatDir="/AstroZone/home/pixel/bcfiles/BeOutFeatDir"

2. irule -F rulesmsiBcExtractFeatureFiles.r "*image='<image>.aff'
   "*outDir='/home/be_feature_dir'"

Files:
- Local File System:
  The following file(s) resides on the Local File System:
  "$iRODS/server/bin/cmd/bulk-extractor"
- iRODS Grid:
Executing this rule creates the following file on the grid:

\$iRODS\_grid/be\_feature\_dir

The actual list of files within this directory depends on the features identified within the image file. Examples:

\$iRODS\_grid/be\_feature\_dir/domain.txt
\$iRODS\_grid/be\_feature\_dir/telephone.txt

**Implementation notes:**

The following file is copied to iRODS/server/bin/cmd directory:

```
cp /usr/local/bin/bulk_extractor iRODS/server/bin/cmd/bulk_extractor
```

5.8.1.3 Generate Annotated Files (identify_filenames)

This tool takes the output files generated by bulk_extractor and the disk image file (.aff or E01 format) as the inputs and creates the annotated versions of each of the feature files generated by the bulk_extractor.

**Input Parameters are:**

- Image File path
- Bulk_extractor directory

**Output Parameter is:**

Output directory annotatedFilesDir to store the annotated files.

```
Tool: identify_filenames --all --imagefile "path/to/imagefile.aff" "Path/to/beFeatDir" "Path/to/outAnnDir"
```

**Command Structure:**

```
irule -F rulemsiBcAnnotateBeFiles.r "*image='/path/to/image.aff'" "*beOutDir='/path/to/beDir'" "*annotateFilesDir='/path/to/newdir'"
```

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/odum-bcAnnotateBeFiles.r.
if(*Num == "0") {
    writeLine("stdout", "Please enter a filename");
    fail;
}

} # Image exists

# Now Make a query to get the path to the image and the resource name
# DATA_PATH: Physical path name for digital object in resource
# DATA_RESC_NAME: Logical name of storage resource
*Query = select DATA_PATH, DATA_RESC_NAME where DATA_NAME = "*File" and COLL_NAME = "*Coll";
foreach (*row in *Query) {
    *PathToImageFile = *row.DATA_PATH;
    *Resource = *row.DATA_RESC_NAME;
    writeLine("stdout", "D: Path to Image file = *PathToImageFile, Resource = *Resource");
}

} # Make another query for IP Address of the resource
# RESC_LOC: Resource IP Address
# DATA_RESC_NAME: Logical name of storage resource
*Query2 = select RESC_LOC where DATA_RESC_NAME = "*Resource";
foreach (*row in *Query2) {
    *Addr = *row.RESC_LOC;
    writeLine("stdout", "D: Host Name = *Addr, Resource = *Resource");
}

} # Query the metadata catalog for the input directory beFeatDir
# First get the Collection ID (CollID_beFeat) of the image. Fail if the
directory doesn't exist
*Q = select count(COLL_NAME) where COLL_NAME = "*beFeatDir";
foreach (*R in *Q) {
    *CollID_beFeat = *R.COLL_NAME;
}
if(int(*CollID_beFeat) == 0) {
    writeLine("stdout", "Please enter a Directory name for BE Features input");
    fail;
}

} # Get the physical path for the directory/collection from DATA_PATH
*Query3 = select DATA_PATH where COLL_ID = "*CollID_beFeat";
foreach (*row3 in *Query3) {
    *FeatFile = *row3.DATA_PATH;

    # The collection name is the same for all the files. So get the
    # collection name for the first one and break out of the loop.
    msiSplitPath(*FeatFile, *Coll, *File);
    writeLine("stdout", "D: Feature Directory: PathToFeatDir = *Coll");
    *PathToFeatDir = *Coll;
    break;
}

dataWriteLine("stdout", "D: Collection Directory = *PathToFeatDir");

} # To have a unique name for the generated file, we will use a prefix
# with timestamp and user-id. The unique file will be generated in /tmp
# directory, which will later be copied to the datagrid.
*prefixStr = "*timeStamp$userNameClient";
*tempStr = "/tmp/*prefixStr" ++ "outAnnDir";

*Arg1 = "‐‐all"
*Arg2 = "‐‐imagefile"
*Arg3 = execCmdArg(*PathToImageFile); # Image:Input
*Arg4 = execCmdArg(*PathToFeatDir); # FeatDir: Input
*Arg5 = execCmdArg(*tempStr); # Output Feature Directory
writeLine("stdout", "Running identify_filenames Command...");
writeLine("stdout", "Command: *Cmd *Arg1 *Arg2 *Arg3 *Arg4 *Arg5");

if (errorcode(msiExecCmd(*Cmd,"*Arg1 *Arg2 *Arg3 *Arg4 *Arg5", "null", "image", "null",*Result)) < 0) {
    if [errormsg("Result","msg")==0] {
        msiGetStderInExecCmdOut("Result",*Out);
        writeLine("stdout", "ERROR: *Out");
    } else {
        writeLine("stdout", "Result msg is empty");
    }
} else {
    # Command executed successfully
    msiGetStdoutInExecCmdOut("Result",*Out);

    # Display the output from the command
    writeLine("stdout", "Command Output: 
    *Out");
    # hintPath is the 4th argument of msiExecCmd, which is a file on the grid
    # The command will be executed on the host where this file is stored.
    *hintPath = "*beFeatDir"++"/report.xml";
    #writeLine("stdout", "D: hintPath: *hintPath");

    # Copy the files from /tmp to the specified output directory
    # Clean up the temporary files
    cleanup(*Addr, *tempStr, *outAnnDir, *prefixStr, *status);
}

# Function: copyFiles()
# copyFiles is called for copying the files in the newly generated directory in
# /tmp to the datagrid.

copyFiles: input string * input string * input string * input string * input integer * input string * output integer
-> integer
    writeLine("stdout", "copyFiles: Moving *tempStr to *outAnnDir");
    *tempStr = *tempStr++"/";
    remote(*Addr, "null") {
        # First make a directory *outAnnDir in the grid by calling a
        # shell script: bcMkdir
        *Arg1 = execCmdArg(*outAnnDir);
        writeLine("stdout", "D: Calling script bcMkdir with arg *Arg1");
        msiExecCmd("bcMkdir", *Arg1, null, null, *hintPath, *Status);

        # Get the list of the annotated files and copy one by one
        # Shell script bcListDir is used to list the files in the
        # temporary directory created in /tmp
        *a1 = execCmdArg(*tempStr);
        msiExecCmd("bcListDir", *a1, null, null, null, *Result);
        msiGetStdoutInExecCmdOut(*Result, *Out);

        # Call split to put the listed files in an array
        *a = split(*Out, ",");
        writeLine("stdout", "D: Files in the array are *a");

        foreach (*file in *a) {
            *file = *tempStr++*file;
            msiSplitPath(*file, *Coll, *just_file);
            *local = "localPath=*tempStr****forceFlag=";
        }
    }
}
# Copy the files from the /tmp/*OutDir location to the grid
*new_outAnnDir = "outAnnDir"++"/"++*just_file;

# Call msiDataObjPut to do the copy of the file
weisLine("stdout", "D: copyFiles: Copying to file new_outAnnDir: *new_outAnnDir");
msiDataObjPut(*new_outAnnDir, "null", *local, *status);
}
}

# Function: cleanup: Calls a script to remove the temporary files created
# in /tmp
cleanup: input string * input string * input string * input string * output integer -> integer
    writeLine("stdout", "Cleanup: Removing temporary files");
    remote(*Addr, "null") {
        *Arg1 = execCmdArg(*prefixStr);
        msiExecCmd("tmpCleanup", *Arg1, "null", "null", "null", *Result);
    }
}

INPUT *image="/AstroZone/home/pixel/bcfiles/charlie-work-usb-2009-12-11.aff",
*beFeatDir="/AstroZone/home/pixel/bcfiles/beFeatDir",
*outAnnDir="/AstroZone/home/pixel/bcfiles/outAnnDir"
OUTPUT ruleExecOut

Command examples:

1. irule -F rulemsiBcAnnotateBeFiles.r
   The default parameters can be modified by changing the following lines appropriately:
   INPUT *image="/AstroZone/home/pixel/bcfiles/charlie-work-usb-2009-12-11.aff",
   *beFeatDir="/AstroZone/home/pixel/bcfiles/beFeatDir",
   *outAnnDir="/AstroZone/home/pixel/bcfiles/outAnnDir"

2. irule -F rulemsiBcAnnotateBeFiles.r "*image='/home/test.aff'"
   "*beOutDir='/home/beDir'" "*annotateFilesDir='/home/annotated_dir'"

Files:

- Local File System:
  The following file(s) resides on the Local File System:
  $iRODS/server/bin/cmd/identify_filenames

- iRODS Grid:
  Executing this rule creates the following file on the grid:
  $iRODS_grid/annotated_dir
  The actual list of files within this directory depends on the features
  identified within the image
  file. Examples:
  $iRODS_grid/annotated_dir/annotated_domain.txt
  $iRODS_grid/annotated_dir/annotated_telephone.txt

Implementation Notes:

The following files are copied to iRODS/server/bin/cmd directory:
~/Research/Tools/bulk_extractor/python/fiwalk.py
~/Research/Tools/bulk_extractor/python/dfxml.py
~/Research/Tools/bulk_extractor/python/bulk_extractor_reader.py
~/Research/Tools/bulk_extractor/python/identify_filenames.py as
identify_filenames

5.8.1.4 Generate BitCurator Reports
This tool takes the xml output of the Fiwalk tool and the annotated files created by identify_filenames as the inputs and produces various reports in Excel and PDF formats in the specified output directory. The Python script is located in irods/server/bin/cmd/bc_generate_reports

Tool: bc_generate_reports

Input Parameters are:
- Annotated Files Directory (Generated by the rule:
  rulemsiBcAnnotateBeFiles.r)
- XML file generated by fiwalk tool (using the rule:
  rulemsiBcGenerateXml.r)

Configuration file
- Output Parameter is:
  Output directory newBcReportsDir where the reports are generated.

Command Structure:

```
irule -F rulemsiBcGenerateReports.r "*fiwalkXmlFile='/Path/To(Xmlfile)'
  *annotatedDir='/Path/To/annotated_directory'
  *outReportsDir='/Path/To/output_Reports_directory'
  *conf='/Path/To/Config_file"
```

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/odum-bcGenerateReportsRule.r.
if (*Num == "0") {
    writeLine("stdout", "Please enter path to XML file");
    fail;
}
}

# hintPath is the 4th argument of msiExecCmd, which is a file on the grid
# The command will be executed on the host where this file is stored. We can use the file
# *fiwalkXmlFile for this purpose as we know it is stored on the grid.
hintPath = "*fiwalkXmlFile";

# XML File exists

# Now Make a query to get the path to the config file and the resource name
# DATA_PATH: Physical path name for digital object in resource
# DATA_RESC_NAME: Logical name of storage resource
*Query = select DATA_PATH, DATA_RESC_NAME where DATA_NAME = "File" and COLL_NAME = "Coll";
foreach (*row in "Query") {
    *PathToXmlFile = *row.DATAPATH;
    *Resource = "row.DATARESC_NAME;"
    writeLine("stdout", "D: Path to XML file = *PathToXmlFile, Resource = *Resource");
}

# Query the metadata catalog for Config File
msiSplitPath("*conf, *Coll, *file1");
Q1 = select count(DATA_NAME) where DATA_NAME = "File1" and COLL_NAME = "Coll1";
foreach (*R1 in "Q1") {
    *Num = "R1.DATANAME;"
    if (*Num == "0") {
        writeLine("stdout", "Please enter path to Config file");
        fail;
    }
}

# Config File exists

# Now Make a query to get the path to the config file and the resource name
# DATA_PATH: Physical path name for digital object in resource
# DATA_RESC_NAME: Logical name of storage resource
*Query_c = select DATA_PATH, DATA_RESC_NAME where DATA_NAME = "File1" and COLL_NAME = "Coll1";
foreach (*row in "Query_c") {
    *PathToConfigFile = "row.DATAPATH;
    *Resource = "row.DATARESC_NAME;"
    writeLine("stdout", "D: Path to Config file = *PathToConfigFile, Resource = *Resource");
}

# Make another query for IP Address of the resource
# RESC_LOC: Resource IP Address
# DATA_RESC_NAME: Logical name of storage resource
*Query2 = select RESC_LOC where DATA_RESC_NAME = "Resource";
foreach (*row in "Query2") {
    *Addr = "row.RESC_LOC;"
    writeLine("stdout", "D: Host Name = *Addr, Resource = *Resource");
}

# Query the metadata catalog for the input directory annotatedDir
# First get the Collection ID (CollID_beFeat) of the Directory. Fail if the
# directory doesn't exist
*Q = select count(COLL_NAME) where COLL_NAME = "annotatedDir";
foreach (*R in "Q") {
    *CollID_annotate = "R.COLL_NAME;"
}
if (*CollID_annotate == 0) {
    writeLine("stdout", "Please enter a Directory name for BE Annotated Files Directory input");
    fail;
}
# Get the physical path for the directory/collection for annotated files directory from DATA_PATH
*Query3 = select DATA_PATH where COLL_ID = "CollID_annotate";

foreach (*row3 in *Query3) {
  *annFile = *row3.DATA_PATH;

  # The collection name is the same for all the files. So get the
  # collection name for the first one and break out of the loop.
  msiSplitPath(*annFile, *Coll, *File);

  writeLine("stdout", "D:\ Feature Directory: annFile: *annFile, PathToAnnDir = *Coll, File=*File");
  *PathToAnnDir = *Coll;
  break;
}

# store the directory of the base collection for future use
msiSplitPath(*Coll, *baseCollDir, "annDir");
writeLine("stdout", "D:\ Collection Dir = *PathToAnnDir");
writeLine("stdout", "D:\ baseCollDir: *baseCollDir");

# To have a unique name for the generated directory, we will use a prefix
# with timestamp and user-id. The unique dir will be generated in /tmp
# directory, which will later be copied to the datagrid.
*prefixStr = "timeStamp$userNameClient";
*tempStr = "/tmp/*prefixStr" ++ "outReportsDir";
*Arg1 = "‐‐fiwalk_xmlfile";
*Arg2 = execCmdArg(*PathToXmlFile);  # XmlFile:Input
*Arg3 = "‐‐annotated_dir";
*Arg4 = execCmdArg(*PathToAnnDir);  # annotatedDir: Input
*Arg5 = "‐‐outdir";
*Arg6 = execCmdArg(*tempStr);  # Output Feature Directory
*Arg7 = "‐‐conf";
*Arg8 = execCmdArg(*PathToConfigFile);
writeLine("stdout", "iRODS: Generating BitCurator Reports...");
writeLine ("stdout", "Command: *Cmd *Arg1 *Arg2 *Arg3 *Arg4 *Arg5 *Arg6 *Arg7 *Arg8");

if (errorcode(msiExecCmd(*Cmd, *Arg1 *Arg2 *Arg3 *Arg4 *Arg5 *Arg6 *Arg7 *Arg8, "null", "null", "null", "null", "null", "null", "null")) < 0) {
  if(errormsg(*Result,*msg)==0) {
    msiGetStderrInExecCmdOut(*Result,*Out);
    writeLine("stdout", "ERROR: *Out");
  } else {
    writeLine("stdout", "Result msg is empty");
  }
} else {
  # Command executed successfully
  msiGetStdoutInExecCmdOut(*Result,*Out);
  writeLine("stdout", "Command Output: \n *Out");

  # Copy the files from /tmp to the specified output directory
  # structure recursively.
  writeLine("stdout", "Copying files from *tempStr to "outReportsDir");

  # Clean up the temporary files
  cleanup(*Addr, *tempStr, *outReportsDir, *prefixStr, *status);
}

# Function: copyFiles()
# copyFiles is called for copying the files in the newly generated directory in
# /tmp to the datagrid.

`copyFiles`: input string * input string * input string * input string * input integer * input string * output integer


  `writeLine("stdout", "copyFiles: Moving *tempStr to *outReportsDir");`

  `remote(*Addr, "null") {`
    # Get the list of the generated report files and copy one by one
    # Shell script bcListFiles is used to list the files in the
    # temporary directory created in /tmp. The files are to be listed in full-path and one per line
    # for the uploadFiles utility function to work.
    # bcListFiles: find $1 -type f -printf "%p
"

    *a1 = execCmdArg(*tempStr);
    msiExecCmd("bcListFiles",*a1, "null", "*hintPath", "null", *Result);

    msiGetStdoutInExecCmdOut(*Result, *Out);

    # Call split to put the listed files in an array
    *a = split(*Out, "\n");

    # writeLine("stdout", D: Files in /tmp: *Out");
    # writeLine("stdout", D: Files in the array are *a");

    *localRoot = *tempStr;

    # Call uploadFiles to copy all the files under *localRoot
    uploadFiles(*localRoot, *a, *outReportsDir)
  }
}

# Function: cleanup: Calls a script to remove the temporary files created
# in /tmp
`cleanup`: input string * input string * input string * input string * output integer -> integer


  `writeLine("stdout", "Cleanup: Removing temporary files");`

  `remote(*Addr, "null") {`
    *Arg1 = execCmdArg(*prefixStr);
    msiExecCmd("tmpCleanup", *Arg1, "null", "null", "null", *Result);
  }
}

INPUT *fiwalkXmlFile="/AstroZone/home/pixel/bcfiles/bcTestFiwalkXmlfile.xml",
*annotatedDir="/AstroZone/home/pixel/bcfiles/bcTestBeAnnDir",
*outReportsDir="/AstroZone/home/pixel/bcfiles/outReportsDir",
*conf="/AstroZone/home/pixel/bcfiles/bcTestConfigFile"

OUTPUT ruleExecOut

Command examples:

1. `irule -F rulemsiBcGenerateReports.r`

   The default parameters can be modified by changing the following line with appropriate parameters:

   ```
   INPUT *fiwalkXmlFile="/AstroZone/home/pixel/bcfiles/bcTestFiwalkXmlfile.xml",
   *annotatedDir="/AstroZone/home/pixel/bcfiles/bcTestBeAnnDir",
   *outReportsDir="/AstroZone/home/pixel/bcfiles/outReportsDir",
   *conf="/AstroZone/home/pixel/bcfiles/bcTestConfigFile"
   ```

2. `irule -F rulemsiBcGenerateReports.r *fiwalkXmlFile="/home/xmlfile"
   "*annotatedDir="/home/annotated_directory"
   "*outReportsDir="/grid/output_directory"
   "*conf="/home/config_file"

Files:
• Local File System:
The following file(s) resides on the Local File System:
$irods/server/bin/cmd/generate_report

• iRODS Grid:
Executing this rule creates the following directories/files on the grid:
$irods_grid/outReportsDir:
$irods_grid/outReportsDir/BeReport.pdf
$irods_grid/outReportsDir/FiwalkDeletedFiles.pdf
$irods_grid/outReportsDir/FiwalkReport.pdf
$irods_grid/outReportsDir/bcTestFiwalkXmlfile.xml.xlsx
$irods_grid/outReportsDir/bc_format_bargraph.pdf
$irods_grid/outReportsDir/format_table.pdf
$irods_grid/outReportsDir/bcfiles/outReportsDir/features
The files under the features directory depends on the image.
Examples are:
$irods_grid/outReportsDir/bcfiles/outReportsDir/features/domain.xlsx
$irods_grid/outReportsDir/bcfiles/outReportsDir/features/telephone.xlsx
$irods_grid/outReportsDir/bcfiles/outReportsDir/features/domain.pdf
$irods_grid/outReportsDir/bcfiles/outReportsDir/features/telephone.pdf

Implementation notes:
The following files are copied to iRODS/server/bin/cmd directory:
$bitcurator/python/bc_reports_tab.py as bc_reports_tab
$bitcurator/python/generate_report.py as bc_generate_reports
$bitcurator/python/bc_utils.py
$bitcurator/python/bc_config.py
$bitcurator/python/bc_pdf.py
$bitcurator/python/bc_graph.py
$bitcurator/python/bc_regress.py
$bitcurator/python/bc_genrep_dfxml.py
$bitcurator/python/bc_genrep_text.py
$bitcurator/python/bc_genrep_xlsx.py
$bitcurator/python/bc_gen_feature_rep_xlsx.py
$bitcurator/python/bc_config_file

5.8.1.5 Bitcurator GUI
BitCurator supports a Graphical User Interface using which users can launch the tools explained above. A rule is written to launch this GUI. But more work needs to be done to make the GUI to appear on the client screen rather than on the server.

Updates to this policy are available from
http://github.com/DICE-UNC/policy-workbook/odum-bcGenerateReportsGuiRule.r
bcGenerateReportsGuiRule {  
# odum-bcGenerateReportsGuiRule.r  
# Command to be executed located in file irods/server/bin/cmd/bc_reports_tab  
# Parameters are:  
# Output: None  
# Input: None  
# Example:  
# irule -F rulesiBcGenerateReportsGui.r  

*Cmd="bc_reports_tab";

writeLine("stdout", "Running Generate Reports command...");
writeLine("stdout", "Command: "+*Cmd");

msiExecCmd(*Cmd, "null", "null", "null", "null", *Result);
if (errorcode(msiExecCmd(*Cmd, "null", "null", "null", "null", *Result)) < 0) {
  if(errormsg(*Result,*msg)==0) {
    msiGetStderrInExecCmdOut(*Result,*Out);
    writeLine("stdout", "ERROR: "+*Out);
  } else {
    writeLine("stdout", "Result msg is empty");
  }
}
msiGetStdoutInExecCmdOut(*Result,*Out);

}INPUT null
OUTPUT ruleExecOut

**Command example:**

*irule -F rulesiBcGenerateReportsGui.r*

### 5.8.1.6 Directory manipulation policy functions

These policy functions were used by the bitcurator policies to manipulate the files in a data grid. They include:

- contains(*list, *elem)
- createCollections(*coll, *cs)
- getFiles(*localRoot, *localPaths)
- getCollections(*filePaths)
- uploadFiles(*localRoot, *localPaths, *coll)

### 5.9 Unique Identifiers for Data Sets (Policy 26)

Multiple external repositories require the generation of a unique data ID. An example is DataONE, which uses the Handle system to assign a unique identifier to a data set. Not all repositories use the same type of identifier. For instance, the California Digital Library uses an ARC identifier.

#### 5.9.1 Assigning a Handle to a File

The Handle system can use a local handle registry for assigning identifiers to files. The local handle registry, in turn, is assigned a unique identifier in a global handle system.

The following rule creates a handle and registers it in the DFC handle server:

(the registration of the handle in our handle server indicates it is available for access
from DataONE)

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acPostProcForPut-handle-nexrad.re.

```plaintext
# rules for creating handle based unique identifiers
# just creating handles for nexrad data right now

acPostProcForPut (ON($userNameClient like "nexrad") {
  msiWriteRodsLog("Attempting to create Handle for $objPath", *Status);
  *Cmd = "create_handle";
  *Keyfile = "/var/lib/irods/hs/admpriv.bin";
  *Uri = "irods%3A%2F%2Firen2.renci.org%3A1237$objPath";
  *Url = "https://dfcweb.datafed.org/idrop-web2/home/link?irodsURL=*Uri";

  *Args = "$dataId $Uri";
  msiExecCmd(*Cmd, *Args, "null", "null", "null", *Result);
  msiGetStdoutInExecCmdOut(*Result, *Out);
  msiWriteRodsLog("Created Handle "$Out" for $objPath", *Status);
})
```

The rule executes a shell script:

```bash
#!/bin/bash

if [ "$#" -ne 2 ]; then
  echo "Usage: create_handle <data object id> <data object url>"
  exit 1
fi

OID="$1"
URL="$2"

HANDLE=$(java -classpath ./irods-hs-tools.jar org.irods.dfc.CreateHandle ./admpriv.bin "$URL" "$OID")

echo "$HANDLE"
exit 0;
```

5.9.2 Registering files in DataONE registry

DataONE web services are used to automate registry of an iRODS collection in the DataONE registry. When the DataONE web service asks for a list of DataONE registered iRODS data objects, the member node web service responds by retrieving the list of objects that have been registered in the handle server. The harvesting is done periodically, with the result that an iRODS data collection can be discovered and accessed through the DataONE services.

5.10 Authentication identity management (Policy 27)

The iRODS data grid provides support for pluggable authentication environments. Each plug-in can also support pre- and post- policy enforcement points. A standard example is the use of an external certificate authority for recognizing users. Any certificate from that certificate authority is honored, and a corresponding user
account is set up in the data grid. Policies control what the new users are allowed to do. This capability was implemented for the Australian Research Collaboration Service.

The iRODS command line tools (icommands) and GridFTP interface can use GSI (Grid Security Infrastructure) authentication which relies on limited lifetime proxy certificates. In addition, your GSI certificate must be mapped to your ARCS Data Fabric account. This is done automatically for ARCS SLCS certificates, and you can add additional mappings for other GSI certificates. A certificate can also be acquired from CILogon through the InCommon infrastructure. An iRODS data grid account can be set up with authentication based on the GSI certificate.

5.10.1 Verify access controls on each file
The data grid manages access control lists for each file. It is possible to query the iCAT catalog to check whether access permission has been given to individuals who should no longer have access. This typically happens when an administrator retires, or the access control policies for a collection have changed. The rule listed in section 4.1.5 identifies access controls on a file in a collection for a specific person.

5.11 Automated Data Reviews (Policy 28)
It is possible to review any of the state information that is stored for a file. A report can be generated which lists all of the non-compliant files within a collection.

5.11.1 Metadata Review
This policy compares the metadata schema that is assigned to a collection with the metadata attributes set on each file within the collection. The collection metadata schema is defined by setting a metadata attribute on the collection with an attribute value of "null".

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/odum-listmetadata.r.
if(int(*Numfiles) > 0) {
    if (int(*Count) > 0) {
        # Loop over the null metadata attributes on the collection
        foreach (*R2 in *Q2) {
            *NameColl = *R2.META_COLL_ATTR_NAME;
        }
        # Loop over the files in the collection
        foreach (*R3 in *Q3) {
            *File = *R3.DATA_NAME;
            *Q4 = select order_asc(META_DATA_ATTR_NAME) where COLL_NAME = "C" and DATA_NAME = "File" and META_DATA_ATTR_UNITS !="iRODSUserTagging:Tag";
            *Q14 = select count(META_DATA_ATTR_NAME) where COLL_NAME = "C" and DATA_NAME = "File" and META_DATA_ATTR_UNITS !="iRODSUserTagging:Tag";
            # Count the number of metadata attributes on a file
            foreach (*R14 in *Q14) {
                *Countf = *R14.META_DATA_ATTR_NAME;
            }
            *Found = 0;
            if (int(*Countf) > 0) {
                # Loop over the tags on a file
                foreach (*R4 in *Q4) {
                    *Name = *R4.META_DATA_ATTR_NAME;
                    if (*NameColl == *Name) {
                        *Found = 1;
                        break;
                    }
                }
            }
            # For missing metadata attributes, print a line to the screen
            if (*Found == 0) {
                writeLine("stdout", "C/*File is missing *NameColl");
            }
        }
    }
}

5.12 Mapping metadata across systems (Policy 29)
The HIVE (Helping Interdisciplinary Vocabulary Engineering) technology is used to integrate vocabularies encoded with the Simple Knowledge Organization System (SKOS), a World Wide Web Consortium (W3C) standard. HIVE is a Linked Open Data (LOD) technology aligning with Linked Open Vocabularies (LOV) activities. The HIVE approach and technologies promote interoperability among data repositories, libraries, and archives, allowing scholarly works to be easily and quickly indexed across multiple disciplines.

The HIVE system can be accessed from the iRODS Data Grid using an updated Curl micro-service. A REST service is available that can query for http://URIs representing concepts in a SKOS vocabulary that is stored in the HIVE system. An example XML representation of a 'concept' in the UAT vocabulary for a given URI is:

```xml
<hiveConcept uri="http://purl.org/astronomy/uat#T100">
    <label>Astroparticle physics</label>
    <altLabel>Particle astrophysics</altLabel>
    <broader uri="http://purl.org/astronomy/uat#T828">
        <label>"Interdisciplinary astronomy"</label>
    </broader>
</hiveConcept>
```
These URIs may be applied to iRODS data objects using the AVU mechanism, where the AVU attribute name is the vocabulary URI, the AVU attribute value is the label, and the AVU unit is a special marker of the form

'iRODSUserTagging:HIVE:VocabularyTerm'

that indicates that the AVU is a resolvable URI.

5.12.1 Validate HIVE vocabularies

An example validation rule utilizes the REST service to iterate over iRODS collections, validating the terms as being valid SKOS references, and generating a report on invalid terms.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/odum-validateOntologies.r.
Here is an example output when two data objects are annotated, one with an invalid term:

```
test1@ubuntu:~/workspace/rule_workbench$ irule -F validate_data_object_ontologies.r
Metadata validation report
/fedZone1/home/rods/hive/libmsiCurlGetObj.cpp has uri
http://purl.org/astronomy/uat#TT888 that is not in a valid ontology
```

### 5.13 Export Datasets in Multiple Formats (Policy 30)

The motivation for changing the format of a file may be to create a standard representation for preservation, or to create a preferred format for display. The ability to export or make available to download datasets in multiple formats such as Excel, CVS, SPSS, or Stata (in other sciences this would include other formats but the issue is the same – being able to go in and out of open and proprietary formats to aid preservation) addresses both future user needs and immediate user needs.

#### 5.13.1 Polyglot Format Conversion

This policy invokes the NCSA Polyglot service to transform a data format. The original file is replaced with the modified file, and metadata attributes are updated. If an attribute named “ConvertMe” is present on the file, the file is converted. The name of the original file is then added as metadata.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acPostProcForModifyAVUMetadata.re.

```
acPostProcForModifyAVUMetadata(*Option,*ItemType,*ItemName,*AName,*AValue,*AUnit) {
   on(*AName == "ConvertMe") {
      irods_curl_get("http://polyglot.cci.drexel.edu/", *ItemName, *AValue, *out);
      if(*out == ""){
         deleteAVUMetadata(*ItemName, "ConvertMe", *AValue, *AUnit, *out3);
         modAVUMetadata(*ItemName, "Conversion Error", *AValue, "dest", *out2);
      }else{
         modAVUMetadata(*out, "Derived from", *ItemName, "iRODS path", *out2);
         deleteAVUMetadata(*ItemName, "ConvertMe", *AValue, *AUnit, *out3);
      }
   }
}
```

### 5.14 Check for viruses (Policy 31)

All files in a staging area can be checked for the presence of a virus. When the check is complete, the files can then be moved into a collection. This uses the clamscan virus check routine which is run as an external executable. The clamscan program must be installed on the iRODS server where the staging area is located in the /usr/bin directory.
5.14.1 Scan files and flag infected objects

This rule runs the clamscan script on an external resource, which checks for the presence of viruses. Each file is flagged with a metadata attribute to record the status of the virus check.

The clamscan python script is:

```python
#!/usr/bin/python
import subprocess, sys
proc = subprocess.Popen(['/usr/local/bin/clamscan'] + sys.argv[1:],
stdout=subprocess.PIPE, stderr=subprocess.STDOUT)
sys.stdout.write(proc.communicate()[0])
sys.stdout.flush()
sys.exit(abs(proc.returncode))
```

The controlling policy can be invoked interactively, or added to the rule base and invoked after each file load.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acScanFileAndFlagObject.re.

```c
acScanFileAndFlagObject(*ObjPath,*FilePath,*Resource)
{
  # Run clamscan script on resource
  # Save operation status
  assign(*Status, $status);
  # Get stdout from last call
  msiGetStdoutInExecCmdOut(*CmdOut, *StdoutStr);

  # Passed, failed, or error?
  if (*Status == 0)
    then
      # Scan successful, object passed
      {
        # Get timestamp
        msiGetSystemTime(*Time, human);
        # Create new KeyValPair_MS_T
        msiAddKeyVal(*KVP, "VIRUS_SCAN_PASSED.*Time", *StdoutStr);
        # Add new metadata triplet to object
        msiAssociateKeyValuePairsToObject(*KVP, *ObjPath, "-d");
      }
    else
      {
        if (*Status == 344000)
          then
            # Scan successful, object failed
            {
              # Get timestamp
              msiGetSystemTime(*Time, human);
              # Create new KeyValPair_MS_T
              msiAddKeyVal(*KVP, "VIRUS_SCAN_FAILED.*Time", *StdoutStr);
              # Add new metadata triplet to object
```
msiAssociateKeyValuePairsToObj(*KVP, *ObjPath, "-d");
}
else
#    # Scan failed (command execution error)
{
    nop;
}
}

5.15 Rule set management (Policy 32)
The iRODS data grid relies upon a distributed rule engine and distributed rule bases to implement policies. If a policy is changed, for consistency the revised rule base needs to be installed at each server location.

5.15.1 Deploy rule sets
This rule identifies the servers, and uploads a new version of the rule base to each server. The micro-services used by this rule are available at https://github.com/DICE-UNC/irods_rule_admin_micorservices

Updates to this policy are available from http://github.com/DICE-UNC/irods-rule-admin-micorservices

# the copy.re must be available on all servers
# the rule admin microservices must also be available on all servers
copyRule {
    # odum-copyRule.r
    remoteDeployRuleSets(*ruleBaseName, *targets);
}

remoteDeployRuleSets(*rbs, *addrs) {
    "out = """
    foreach(*addr in *addrs) {
        foreach(*rb in *rbs) {
            *err = errorcode(remoteWriteRuleSet(*rb, *addr));
            *out = *out ++ "*rb -> *addr " ++ (if *err != 0 then "failure" else "success") ++ "\n";
        }
    }
    writeLine("stdout", *out);
}

remoteWriteRuleSet(*rb, *addr) {
    msiReadRuleSet(*rb, *rule);
    msiChksumRuleSet(*rb, *chksum);
    remote(*addr, "") {
        writeRuleSet(*rb, *rule, *chksum);
    }
}
writeRuleSet(*rb, *rule, *chksum) {
    backupRuleSet(*rb, *rbak):::
        if(errorcode(*rbak) == 0) {
            msiMvRuleSet(*rbak, *rb);
        }
5.16 Parse event trail for all persons accessing a collection (Policy 33)

The DFC DataBook system provides a way to record information about events that occur on files within the data grid. This policy is implemented in the rule base, such that events are automatically tracked across all clients. The policies are available in the file iRODS/server/config/reConfigs/databook.re.

```
include("path = (\*path like regex ".*/\[/\].preview(\.[^/]+")\)"
ATTR_ID = "data:id"
ATTR_NAME = "data:name"
ATTR_DATA_SIZE = "data:size"
ATTR_THUMB_PREVIEW = "data:thumbPreview"
ATTR_PREVIEW = "data:preview"
ATTR_HAS_VERSION = "data:hasVersion"
ATTR_TITILE = "data:title"
ATTR_DESCRIPTION = "data:description"
ATTR_REPLACES = "data:replaces"
ATTR_REPLACED_BY = "data:replacedBy"
ATTR_RELATED = "data:related"
ATTR_CONTRIBUTOR = "data:contributor"
ERROR_CODE = -1

# AVU Attr's that are mapped to Databook Attr's
# Do not include system attributes such as data:name, data:dataSize, or data:id
databookAttr(*Attr) =
```
match *Attr with
| ATTR_ID => true
| ATTR_HAS_VERSION => true
| ATTR_PREVIEW => true
| ATTR_THUMB_PREVIEW => true
| ATTR_CONTRIBUTOR => true
| ATTR_RELATED => true
| ATTR_REPLACED_BY => true
| ATTR_REPLACES => true
| ATTR_TITLE => true
| ATTR_DESCRIPTION => true
| _ => false

databookAttrType(*Attr) =
match *Attr with
| ATTR_ID => "string"
| ATTR_HAS_VERSION => "string"
| ATTR_PREVIEW => "string"
| ATTR_THUMB_PREVIEW => "string"
| ATTR_CONTRIBUTOR => "string"
| ATTR_RELATED => "string"
| ATTR_REPLACED_BY => "string"
| ATTR_REPLACES => "string"
| ATTR_TITLE => "string"
| ATTR_DESCRIPTION => "string"
| _ => let *dummy = writeLine("serverLog", "databookAttrType: unsupported data attribute *Attr, return default type 'string'") in "string"
constant(*Attr) =
match *Attr with
| ATTR_ID => true
| ATTR_HAS_VERSION => false
| ATTR_PREVIEW => false
| ATTR_THUMB_PREVIEW => false
| ATTR_CONTRIBUTOR => false
| ATTR_RELATED => false
| ATTR_REPLACED_BY => false
| ATTR_REPLACES => false
| ATTR_TITLE => false
| ATTR_DESCRIPTION => false
| _ => false
unique(*Attr) =
match *Attr with
| ATTR_ID => true
| ATTR_HAS_VERSION => true
| ATTR_PREVIEW => false
| ATTR_THUMB_PREVIEW => false
| ATTR_CONTRIBUTOR => false
| ATTR_RELATED => false
| ATTR_REPLACED_BY => false
| ATTR_REPLACES => false
| ATTR_TITLE => true
| ATTR_DESCRIPTION => true
| _ => false
required(*Attr) =
match *Attr with
| ATTR_ID => true
| ATTR_HAS_VERSION => true
| ATTR_PREVIEW => false
| ATTR_THUMB_PREVIEW => false
| ATTR_CONTRIBUTOR => false
| ATTR_RELATED => false
| ATTR_REPLACED_BY => false
| ATTR_REPLACES => false
| ATTR_TITLE => false
| ATTR_DESCRIPTION => false
| _ => false

timeStr(*time) = timeStrf("time, "%Y %m %d %H:%M:%S")
timeStrNow = timeStr(time())

acPostProcForPut {
  on(include($objPath)) {
    postProcForCreateCommon($objPath, $dataSize, true);
    # preview must be generated after the file is uploaded
    genPreviewGen($objPath, *previewThumbPath, *previewPath);
  }
}

acPostProcForCopy {
  on(include($objPath)) {
    postProcForCreateCommon($objPath, $dataSize, true);
    genPreviewGen($objPath, *previewThumbPath, *previewPath);
    #
    # *sourceAssociatedFiles = getAssociatedFiles($srcObjPath);
    # *destAssociatedFiles = getAssociatedFiles($objPath);
    # both list should have the same size
    #
    # *size = size(*sourceAssociatedFiles);
    #
    #
    # for(*i=0; *i<*size; *i=*i+1) {
    #  writeLine("serverLog", "copying "++elem(*sourceAssociatedFiles, *i)++" to "++elem(*destAssociatedFiles, *i));
    #  msiDataObjCopy(elem(*sourceAssociatedFiles, *i), elem(*destAssociatedFiles, *i), "forceFlag", *Status);
    #}
    
    # avoid write back as output param
    postProcForCreateCommon: input string *input f input double * boolean -> integer
    postProcForCreateCommon(*objPath, *dataSize, *put) {
      *dataId = _getDataObjId(*objPath, *Found);
      if(*Found) {
        sendUpdateDataObj($userNameClient, *objPath, str(*dataSize), *dataId, *put);
      } else {
        genDataObjId(*objPath, *dataId);
        (*collName, *dataName) = splitDataObjPath(*objPath);
        *createTime = getFirstResult(
          SELECT DATA_CREATE_TIME WHERE DATA_NAME = *dataName AND COLL_NAME = *collName,
          "DATA_CREATE_TIME",
          *Found
        );
        *time = datetime(double(*createTime));
        *formattedTimeStr = timeStr(*time);
        writeLine("serverLog", *postProcForCreateCommon: new data obj *objPath, *dataSize, *put);
        sendAddDataObj($userNameClient, *objPath, str(*dataSize), *formattedTimeStr, *put);
      }
    }
}

acPostProcForDelete {
  on(include($objPath)) {
    sendRemoveDataObj($userNameClient, $objPath);
    foreach(*filename in getAssociatedFiles($objPath)) {
      msiDataObjUnlink(*filename, *Status);
    }
  }
}

acPostProcForRename(*sourceObject,*destObject) {
  on(include(*sourceObject)) {
    getFirstResult{
      SELECT COLL_NAME WHERE COLL_NAME = (str(*destObject)),
      "COLL_NAME",
      *found
    );
    if(*found) {
      sendMoveColl($userNameClient, *sourceObject, *destObject);
    }
  }
}
```c
} else {
    sendMoveDataObj($userNameClient, *sourceObject, *destObject);
    *sourceAssociatedFiles = getAssociatedFiles(*sourceObject);
    *destAssociatedFiles = getAssociatedFiles(*destObject);
    # both list should have the same size
    *size = size(*sourceAssociatedFiles);
    for(*i=0; *i< size; *i++) {
        msiDataObjRename(elem(*sourceAssociatedFiles, *i), elem(*destAssociatedFiles, *i), 0, *Status);
    }
}

acPostProcForCollCreate{
    on(include($collName)) {
        genCollId($collName, *Id);
        sendAddColl($userNameClient, $collName);
        # or {
        #    writeLine("serverLog", "ignored collect $objPath");
        # }
    }
}

acPreprocForRmColl{
    on(include($collName)) {
        writeLine("serverLog", "acPreprocForRmColl: $collName");
        sendRemoveColl($userNameClient, $collName);
    }
}

acPostProcForOpen{
    on(include($objPath)) {
        writeLine("serverLog", "acPostProcForOpen: $objPath, writeFlag = $writeFlag, dataSize = $dataSize");
        *dataId = getDataObjId($objPath);
        if($writeFlag=="0") {
            # this is probably a get operation
            sendGetDataObj($userNameClient, $objPath);
        } else {
            sendUpdateDataObj($userNameClient, $objPath, str($dataSize), getDataObjId($objPath), false);
        }
        #sendAccessDataObj(ACCESS_TYPE_DATA_OBJ_CLOSE, $userNameClient, $objPath, *dataId);
    }
}

acPostProcForModifyCollMeta {}

acPostProcForModifyDataObjMeta {}

acPreProcForModifyAVUMetadata(Option, *ItemType, *SrcItemName, *TgtItemName) {
    on(include("*ItemName")) {
        preProcForModifyAVUMetadata(Option, *ItemType, *SrcItemName, *TgtItemName, "", "", "", "", "");
    }
}

acPreProcForModifyAVUMetadata(Option, *ItemType, *ItemName, *AName, *AValue, *AUnit) {
    on(include("*ItemName")) {
        preProcForModifyAVUMetadata(Option, *ItemType, *ItemName, *AName, *AValue, *AUnit, "", "", "");
    }
}

acPreProcForModifyAVUMetadata(Option,*ItemType,*ItemName,*AName,*AValue, *AUnit, *NAName,*NAValue,*NAUnit) {
    on(include("*ItemName")) {
        preProcForModifyAVUMetadata(Option, *ItemType, *ItemName, *AName, *AValue, *AUnit, *NAName, *NAVvalue, *NAUnit);
    }
}
```
checkMultiplicity(*AName,*objPath,*min,*max) {
  *ret = true;
  (*coll,*data) = splitDataObjPath(*objPath);
  foreach[*r in SELECT count(META_DATA_ATTR_NAME) WHERE META_DATA_ATTR_NAME = *AName AND COLL_NAME = *coll AND DATA_NAME = *data] {
    *mul = int[*r.META_DATA_ATTR_NAME]
    if((*min >= 0 && *mul < *min) || (*max >= 0 && *mul > *max)) {
      *ret = false;
    }
    break;
  }
  *ret;
}

preProcForModifyAVUMetadata(*Option,*ItemType,*ItemName,*AName,*AValue,*AUnit,*NAName,*NAValue,*NAUnit) {
  on(databookAttr(*AName)) {
    cut;
    if(constant(*AName) && *Option != "add") {
      failmsg(-1, "cannot perform operation *Option on attribute *AName");
    }
    if("Option like "mod" && *NAName != ") {
      failmsg(-1, "cannot modify name for attribute *AName");
    }
    if("Option like "add" && unique(*AName)) {
      if(!checkMultiplicity(*AName, *ItemName, 0, 0)) {
        failmsg(-1, "cannot have more than one value for attribute *AName");
      }
    }
    if("Option like "rm" && required(*AName)) {
      if(!checkMultiplicity(*AName, *ItemName, 2, -1)) {
        failmsg(-1, "cannot have more than one value for attribute *AName");
      }
    }
    or {
      succeed;
    }
  }
}

acPostProcForModifyAVUMetadata(*Option,*ItemType,*SrcItemName,*TgtItemName) {
  on(include(*ItemName)) {
    postProcForModifyAVUMetadata(*Option,*ItemType,*SrcItemName,*TgtItemName,"","","","","","");
  }
}

acPostProcForModifyAVUMetadata(*Option,*ItemType,*ItemName,*AName,*AValue,*AUnit) {
  on(include(*ItemName)) {
    postProcForModifyAVUMetadata(*Option,*ItemType,*ItemName,*AName,*AValue,*AUnit,"","","","","","");
  }
}

acPostProcForModifyAVUMetadata(*Option,*ItemType,*ItemName,*AName,*AValue,*AUnit,*NAName,*NAValue,*NAUnit) {
  on(include(*ItemName)) {
    postProcForModifyAVUMetadata(*Option,*ItemType,*ItemName,*AName,*AValue,*AUnit,*NAName,*NAValue,*NAUnit);
  }
}

postProcForModifyAVUMetadata(*Option,*ItemType,*ItemName,*AName,*AValue,*AUnit,*NAName,*NAValue,*NAUnit) {
  on(*AName == "data:id") {
    writeLine(" serverLog","skip data:id");
    succeed;
  }
  on(databookAttr(*AName)) {
    cut;
  }
}
writeLine("serverLog", "processing *AName for *ItemName");
if("Option == "rmi")
{
    if("ItemType == "-d")
    {
        *Id = getDataObjIdById(*ItemName);
    }
    else if("ItemType == "-C")
    {
        *Id = getCollIdById(*ItemName);
    }
    else {
        failmsg(-1, "unsupported object type: *ItemName, *ItemType");
    }
}
else {
    if("ItemType == "-d")
    {
        *Id = getDataObjId(*ItemName);
    }
    else if("ItemType == "-C")
    {
        *Id = getCollId(*ItemName);
    }
    else {
        failmsg(-1, "unsupported object type: *ItemName, *ItemType");
    }
}

# need to check *AVal format
*DatabookName=triml(*AName, ":");
writeLine("serverLog", "*Option");
if("Option like "add*")
{
    *
    *msg=join(list("addMeta", *Id, *DatabookName, *AValue, databookAttrType(*AName)));
    *accessType=ACCESS_TYPE_METADATA_ADD;
}
else if("Option like "rm*")
{
    *msg=join(list("delMeta", *Id, *DatabookName, *AValue, databookAttrType(*AName)));
    *accessType=ACCESS_TYPE_METADATA_REMOVE;
}
else if("Option like "set*")
{
    *msg=join(list("modMeta", *Id, *DatabookName, *AValue, databookAttrType(*AName)));
    *accessType=ACCESS_TYPE_METADATA_MODIFY;
}
else if("Option like "mod*")
{
    *msg=join(list("modMeta", *Id, *DatabookName, *NAValue, databookAttrType(*AName)));
    *accessType=ACCESS_TYPE_METADATA_MODIFY;
}
else {
    failmsg(-1, "unsupported option: "Option, *ItemName");
}
amqpSend("localhost", "metaQueue", *msg);
# send access log
sendAccess(*accessType, *userNameClient, *Id, timeStrNow(), "*AName *accessType");
or{
    writeLine("serverLog", "unrecognized metadata attribute: *ItemName, *AName, *AValue, *AUnit");
}
}

sendAddDataObj(*userName, *objPath, *dataSize, *dateCreated, *put) {
    *coll=trimr(str(*objPath), "/");
    *collId=getCollId(*coll);
    *objId=getDataObjId(*objPath);
    *msg=join(list("add", *objId, str(*objPath), DataObject, *collId));
amqpSend("localhost", "metaQueue", *msg);
    amqpSend("localhost", "metaQueue", *msg);
    *msg=join(list("addMeta", *objId, Created, *dataSize, integer));
amqpSend("localhost", "metaQueue", *msg);
    *msg=join(list("addMeta", *objId, Created, *dateCreated, dateTime));
amqpSend("localhost", "metaQueue", *msg);
    *msg=join(list("addMeta", *objId, Owner, *userName, object));
amqpSend("localhost", "metaQueue", *msg);
    # send access log
    sendAccessDataObj(if *put then ACCESS_TYPE_DATA_OBJ_PUT else ACCESS_TYPE_DATA_OBJ_CREATE, *userName, *objPath, *objId);
}

sendAddColl(*userName, *collPath) {
    *coll=trimr(str(*collPath), "/");
    if("coll==" "")
    {
        *collId="root";
    }
    else {
        *collId=getCollId(*coll);
    }
}
*objId=getCollId("collPath");
*msg=join(list("add", *objId, str("collPath"); Collection", *collId));
#writeLine("stdout", message = *msg);
amqpSend("localhost", "metaQueue", *msg);
# send access log
sendAccessDataObj(ACCESS_TYPE_COLL_CREATE, *userName, str("collPath"); *objId);
}
sendUpdateDataObj(*userName, *objPath, *dataId, *put) {
 *msg=join(list("modMeta", *dataId, "dataSize", *dataSize, "integer"));
amqpSend("localhost", "metaQueue", *msg);
# send access log
sendAccessDataObj(if *put then ACCESS_TYPE_DATA_OBJ_OVERWRITE else ACCESS_TYPE_DATA_OBJ_UPDATE,
*userName, *objPath, *dataId);
}
sendRemoveDataObj(*userName, *objPath) {
 *coll=trimr(*objPath, "/");
 *collId=getCollId(*coll);
 *objId=getDataObjId(*objPath);
 *msg=join(list("del", *objId, *collId));
amqpSend("localhost", "metaQueue", *msg);
# send access log
sendAccessDataObj(ACCESS_TYPE_DATA_OBJ_REMOVE, *userName, *objPath, *objId);
}
sendRemoveColl(*userName, *objPath) {
 *coll=trimr(*objPath, "/");
 *collId=getCollId(*coll);
 writeLine("serverLog", "collId = *collId");
 *objId=getCollId(*objPath);
 writeLine("serverLog", "objId = *objId");
 *msg=join(list("del", *objId, *collId));
amqpSend("localhost", "metaQueue", *msg);
# send access log
sendAccessDataObj(ACCESS_TYPE_COLL_DELETE, *userName, *objPath, *objId);
}
sendGetDataObj(*userName, *objPath) {
 *objId=getDataObjId(*objPath);
# send access log
sendAccessDataObj(ACCESS_TYPE_DATA_OBJ_GET, *userName, *objPath, *objId);
}
sendMoveDataObj(*userName, *objPathOld, *objPathNew) {
 *collOld=trimr("objPathOld", "/");
 *collNew=trimr("objPathNew", "/");
 *collOldId=getCollId(*collOld);
 *collNewId=getCollId(*collNew);
 *objId=getDataObjId(*objPathNew);
 *msg=join(list("move", *objId, "objPathOld", *objPathNew, *collOldId, *collNewId));
amqpSend("localhost", "metaQueue", *msg);
# send access log
sendAccessDataObj(ACCESS_TYPE_DATA_OBJ_MOVE, *userName, *objPathOld, *objPathNew, *objId);
}
sendMoveColl(*userName, *objPathOld, *objPathNew) {
 *collOld=trimr("objPathOld", "/");
 *collNew=trimr("objPathNew", "/");
 *collOldId=getCollId(*collOld);
 *collNewId=getCollId(*collNew);
 *objId=getCollId(*objPathNew);
 *msg=join(list("move", *objId, "objPathOld", *objPathNew, *collOldId, *collNewId));
amqpSend("localhost", "metaQueue", *msg);
# send access log
sendAccessDataObj(ACCESS_TYPE_COLL_MOVE, *userName, *objPathOld, *objPathNew, *objId);
}
sendCollectionStructure(*path) {
*dataObjs = SELECT DATA_NAME, COLL_NAME, DATA_SIZE WHERE COLL_NAME like "*path%";
foreach(*dataObj in *dataObjs) {
    writeLine(stdout, "*dataObj");
    *dataName = *dataObj.DATA_NAME;
    *dataSize = *dataObj.DATA_SIZE;
    *collName = *dataObj.COLL_NAME;
    *objPath = /*collName/*dataName;
    sendAddDataObj("sync", *objPath, *dataSize, timeStrNow(), false);
}

*dataObjs = SELECT COLL_NAME WHERE COLL_NAME like "*path%";
foreach(*dataObj in *dataObjs) {
    *objPath = path(*dataObj.COLL_NAME);
    sendAddColl("sync", *objPath);
}

# access type:
# automatic:
ACCESS_TYPE_DATA_OBJ_MOVE = "data obj move"
ACCESS_TYPE_DATA_OBJ_REMOVE = "data obj remove" # remove is issued automatically from peps
ACCESS_TYPE_COLL_MOVE = "coll move"
ACCESS_TYPE_DATA_OBJ_UPDATE = "data object update" # write close
ACCESS_TYPE_DATA_OBJ_OVERWRITE = "data object overwrite" # overwriting put/copy
ACCESS_TYPE_DATA_OBJ_PUT = "data object put" # used only for non overwriting put/copy, a combination of
CREATE, WRITE, UPDATE
ACCESS_TYPE_DATA_OBJ_GET = "data object get" # read-only close
ACCESS_TYPE_METADATA_ADD = "metadata add"
ACCESS_TYPE_METADATA_REMOVE = "metadata remove"
ACCESS_TYPE_METADATA_MODIFY = "metadata modify"
# manual:
ACCESS_TYPE_RULE = "rule"
ACCESS_TYPE_MICROSERVICE = "microservice"
ACCESS_TYPE_DATA_OBJ_READ = "data obj read"
ACCESS_TYPE_DATA_OBJ_WRITE = "data obj write"
ACCESS_TYPE_DATA_OBJ_CLOSE = "data obj close" # current this is unused, either an UPDATE, PUT, or GET is used instead
ACCESS_TYPE_DATA_OBJ_CREATE = "data obj create"
ACCESS_TYPE_DATA_OBJ_DELETE = "data obj delete" # current this is unused, when a data obj is moved to trash,
MOVE is used, and when deleted from trash, REMOVE is used
ACCESS_TYPE_COLL_READ = "coll read"
ACCESS_TYPE_COLL_WRITE = "coll write"
ACCESS_TYPE_COLL_CREATE = "coll create"
ACCESS_TYPE_COLL_DELETE = "coll delete"
sendAccess(*AccessType, *UserName, *DataId, *Time, *Description) {
    sendAccessWithSession(*AccessType, *UserName, *DataId, *Time, *Description, getGlobalSessionId());
}

sendAccessWithSession(*AccessType, *UserName, *DataId, *Time, *Description, *SessionId) {
    *AccessId = genAccessId(*AccessType, *UserName, *DataId, *Time, *Description);
    *msg = join(list("access", *DataId, *AccessId, *AccessType, *UserName, *Time, *Description, *SessionId));
    amqpSend("localhost", "metaQueue", *msg);
}

}

    *AccessId = genAccessId(*AccessType, *UserName, *ActionId, *TimeStart, *Description);
    amqpSend("localhost", "metaQueue", *msg);
}

splitDataObjPath(*objPath) {
    *objPathStr = str(*objPath);
}
*objPathStrLen = strlen(*objPathStr);
*collName = trimr(*objPathStr, "/");
*dataNameLen = *objPathStrLen - strlen(*collName) - 1;
*dataName = substr(*objPathStr, *objPathStrLen - *dataNameLen, *objPathStrLen);
(*collName, *dataName);
}

dataObjIdById(*id) {
    foreach(*r in SELECT DATA_NAME, COLL_NAME WHERE DATA_ID = *id) {
        *objPath = r.COLL_NAME ++ "/" ++ r.DATA_NAME;
    break;
    }
    getDataObjId(*objPath, *Found);
}

collIdById(*id) {
    foreach(*r in SELECT COLL_NAME WHERE COLL_ID = *id) {
        *collPath = r.COLL_NAME;
    break;
    }
    getCollId(*collPath, *Found);
}

collId(*coll) {
    writeLine(”serverLog”, ”getCollId of *coll”);
    *Id = getFirstResult(SELECT META_COLL_ATTR_VALUE WHERE COLL_NAME = (str(*coll)) AND META_COLL_ATTR_NAME = ATTR_ID,
        ”META_COLL_ATTR_VALUE”,
        *Found
    );
    if(!*Found) {
        genCollId(*coll, *Id);
    }
    writeLine(”serverLog”, *Id);
    *Id;
}

dataObjId(*objPath, *Found) {
    (*collName, *dataName) = splitDataObjPath(*objPath);
    writeLine(”serverLog”, ”getObjId of *objPath, *dataName, *collName”);
    *Id = getFirstResult(SELECT META_DATA_ATTR_VALUE WHERE DATA_NAME = *dataName AND COLL_NAME = *collName AND META_DATA_ATTR_NAME = ATTR_ID,
        ”META_DATA_ATTR_VALUE”,
        *Found
    );
    *Id;
}

dataObjId(*objPath) {
    *Id = _getDataObjId(*objPath, *Found);
    if(!*Found) {
        genDataObjId(*objPath, *Id);
    }
    writeLine(”serverLog”, *Id);
    *Id;
}

dataObjId(*path, *Id) {
    *Id = str(*path) ++ str(double(time()));
    addMetaAV(*path, ATTR_ID, *Id, ”-d”);
}

collId(*path, *Id) {
    # writeLine(”serverLog”, ”genCollId: *path”);
    *Id = str(*path) ++ str(double(time()));
}
addMetaAV("path, ATTR_ID, "Id", ",-C");
}
genAccessId(*AccessType, *UserName, *DataId, *Time, *Description) {
	# need a more sophisticated method
	AccessId = DataId++"a"++str(double(time()))++replace("AccessType, ", ",");
	AccessId;
}

genFileType(*objPath) {
	(*collName, *dataName) = splitDataObjPath(*objPath);
	phyPath = getPhyPath(*collName, *dataName);
	writeLine("serverLog", genFileType: *objPath, *phyPath);
	e = execCmd("file", list("-b", *phyPath), *status);
	if(*e < 0) {
		*t = ";
	} else {
		msiGetStdoutInExecCmdOut(*status, *out);
		*l = strstr(*out, ");
		if(*l == -1) {
			*t = ";
		} else {
			*t = substr(*out, 0, *");
		}
	}
	*t;
}
canGenPreview(*objPath) =
let *ext = genFileType(*objPath) in
*ext == "JPEG" || *ext == "PNG" || *ext == "GIF" || *ext == "PDF"
genPreviewGen(*objPath, *previewThumbPath, *previewPath) {
	on(getFileType(*objPath)=="JPEG") {
		picPreview(*objPath, *previewThumbPath, *previewPath);
	}
	on(getFileType(*objPath)=="PNG") {
		picPreview(*objPath, *previewThumbPath, *previewPath);
	}
	on(getFileType(*objPath)=="GIF") {
		pdfPreview(*objPath, *previewThumbPath, *previewPath);
	}
	on(getFileType(*objPath)=="PDF") {
		pdfPreview(*objPath, *previewThumbPath, *previewPath);
	}
	or {
		writeLine("serverLog", genPreview: unsupported file type for *objPath, ext = "++getFileType(*objPath));
		*previewPath = "; # set preview path to empty
	}
}

join(*list) {
	*out = ""

tforeach(*a in *list) {
	*out = "out"+*a+"n";
}

substr(*out, 0, strlen(*out)-1);
}

getFileExt(*objPath) =
let *baselen = strlen(trimr(*objPath, ")") in
let *objPathLen = strlen(*objPath) in
if *baselen == *objPathLen then "" else substr(*objPath, *baselen+1, *objPathLen)

getAssociatedFiles(*objPath) =
let (*collName, *dataName) = splitDataObjPath(*objPath) in
let *p = getPhyPath(*collName, *dataName) in
if canGenPreview(*objPath) then
	let (*previewPath, *previewPathTmp, *previewThumbPath, *previewThumbPathTmp) =
getPreviewPaths(*collName, *dataName) in
    list(*previewPath, *previewThumbPath)
else
    list()

getPreviewPaths(*collName, *dataName) =
    let *timeStr = str(double(time())) in
    (str('/' + *collName + '.preview/' + *dataName + '.preview.jpeg'),
     str('/tmp/' + *dataName + '.timeStr.preview.jpeg'),
     str('/' + *collName + '.preview/' + *dataName + '.previewThumb.jpeg'),
     str('/tmp/' + *dataName + '.timeStr.previewThumb.jpeg'))

getPhyPath(*collName, *dataName) =
    getFirstResult(
        SELECT DATA_PATH WHERE DATA_NAME = *dataName AND COLL_NAME = *collName,
        "DATA_PATH",
        "Found"
    )

genPreview(*objPath, *previewThumbPath, *previewPath, *script) {
    (*collName, *dataName) = splitDataObjPath(*objPath);
    *previewCollection = /*collName/.preview;
    createCollIfNotExist(*previewCollection);
    *objPhyPath = getPhyPath(*collName, *dataName);
        getPreviewPaths(*collName, *dataName);
    writeLine("serverLog", "generating preview for *objPath, ext = " + getFileExt(*objPath) + ", phypath = *objPhyPath");
    *errcode = execCmd(*script, list(*objPhyPath, *previewThumbPath, *previewThumbPathTmp, *previewPath,
            *previewPathTmp), *status);
    if(*errcode < 0) {
        writeLine("serverLog", "convert thumbnail error *errcode"); # : out = *out, err = *err);
    }
    # send thumb link first, the preview attribute triggers a caching on the VIVO side
    addMetaAV(*objPath, ATTR_THUMB_PREVIEW, "-d");
    addMetaAV(*objPath, ATTR_PREVIEW, "-d");
}

genPicPreview(*objPath, *previewThumbPath, *previewPath) {
    genPreview(*objPath, *previewThumbPath, *previewPath, "convertThumbnail.sh")
}

genPdfPreview(*objPath, *previewThumbPath, *previewPath) {
    genPreview(*objPath, *previewThumbPath, *previewPath, "convertPdfThumbnail.sh")
}

createCollIfNotExist(*path) {
    getFirstResult(
        SELECT COLL_NAME WHERE COLL_NAME = (str(*path)),
        "COLL_NAME",
        "found"
    );
    if(! *found) {
        msiCollCreate(*path, "0", *status);
    }
}

getKVP(*KVP, *Key) {
    msiGetValByKey(*KVP, *Key, *Val);
    *Val;
}
getFirstResult(*rs, *column, *found) {
    # dummy code to force the type of *RS
    if(false) {
        *rs = SELECT DATA_PATH WHERE DATA_NAME="rods";
    }
    # writeLine("serverLog", "get first result");
    # writeLine("serverLog", type(*rs));
    *found = false;
    *id = ""
    foreach(*a in *rs) {
        *id = getKVP(*a, *column);
        *found = true;
        break;
    }
    *id;
}

addMetaAV(*objPath, *attr, *value, *type) {
    msiString2KeyValPair("*attr=*value", *kvp);
    msiAssociateKeyValuePairsToObj(*kvp, *objPath, *type);
    *databookName=triml(*attr, ":");
    if(*type == ":-d") {
        *id = getDataObjId(*objPath);
    } else if(*type == ":-C") {
        *id = getCollId(*objPath);
    }
    *msg=join(list("addMeta", *id, *databookName, *value, databookAttrType(*attr)));
    amqpSend("localhost", "metaQueue", *msg);
}

execCmd(*cmd, *args, *status) {
    *argStr = ""
    foreach(*arg in *args) {
        *argStr = "*argStr++execCmdArg(*arg)++";
    }
    *argStr = substr(*argStr, 0, strlen(*argStr) - 1);
    writeLine("serverLog", *argStr);
    *e = errorCode(msiExecCmd(*cmd, *argStr, "null", "null", "null", *status));
    if(*e < 0) {
        msiGetStderrInExecCmdOut(*status, *err);
        writeLine("serverLog", *err);
    }
    *e;
}

# wrapper rules for data access
sendAccessDataObjCurrentUser(*accessType, *objPath, *dataId) {
    sendAccessDataObj(*accessType, *userNameClient, *objPath, *dataId);
}

sendAccessDataObj(*accessType, *userId, *objPath, *dataId) {
    if(*accessType == ACCESS_TYPE_DATA_OBJ_CREATE) {
        *description = "<img src='images/icons/create.png' alt='*objPath'>";
    } else if(*accessType == ACCESS_TYPE_DATA_OBJ_OPEN) {
        *description = "<img src='images/icons/open.png' alt='*objPath'>";
    } else if(*accessType == ACCESS_TYPE_DATA_OBJ_READ) {
        *description = "<img src='images/icons/read.png' alt='*objPath'>";
    } else if(*accessType == ACCESS_TYPE_DATA_OBJ_WRITE) {
        *description = "<img src='images/icons/write.png' alt='*objPath'>";
    } else if(*accessType == ACCESS_TYPE_DATA_OBJ_CLOSE) {
        *description = "<img src='images/icons/close.png' alt='*objPath'>";
    } else if(*accessType == ACCESS_TYPE_DATA_OBJ_DELETE) {
        *description = "<img src='images/icons/delete.png' alt='*objPath'>";
    } else if(*accessType == ACCESS_TYPE_DATA_OBJ_UPDATE) {
        *description = "<img src='images/icons/update.png' alt='*objPath'>";
    }
}
} else if(*accessType == ACCESS_TYPE_DATA_OBJ_OVERWRITE) {
    *description = "<img src='images/icons/overwrite.png'/>*objPath; 
} else if(*accessType == ACCESS_TYPE_DATA_OBJ_PUT) {
    *description = "<img src='images/icons/put.png'/>*objPath; 
} else if(*accessType == ACCESS_TYPE_DATA_OBJ_GET) {
    *description = "<img src='images/icons/get.png'/>*objPath; 
} else if(*accessType == ACCESS_TYPE_DATA_OBJ_MOVE) {
    # split objPath
    (*objPathOld, *objPathNew) = *objPath;
    *description = "*objPathOld<img src='images/icons/move.png'/>*objPathNew;"
} else if(*accessType == ACCESS_TYPE_COLL_MOVE) {
    # split objPath
    (*objPathOld, *objPathNew) = *objPath;
    *description = "*objPathOld<img src='images/icons/move.png'/>*objPathNew;"
} else {
    *description = "*objPath *accessType;"
} sendAccess(*accessType, *userId, *dataId, timeStrNow(), *description);
}

# this closes the data obj immediately to trigger event
dataObjCreate(*objPath, *options, *dataObj) {
    msiDataObjCreate(*objPath, *options, *desc);
    postProcForCreateCommon(*objPath, 0, false);
    *dataId = getDataObjId(*objPath);
    *dataObj = (*desc, *objPath, *dataId);
}
dataObjLseek(*dataObj, *off, *base, *status) {
    (*desc, *objPath, *dataId) = *dataObj;
    msiDataObjLseek(*desc, *off, *base, *status);
}
dataObjOpen(*objPath, *options, *dataObj) {
    if (*options == "") {
        *kvpStr = *objPath;
    } else {
        *kvpStr = "objPath=*objPath++++*options";
    }
    msiDataObjOpen(*kvpStr, *desc);
    *dataId = getDataObjId(*objPath);
    *dataObj = (*desc, *objPath, *dataId);
    sendAccessDataObj(ACCESS_TYPE_DATA_OBJ_OPEN, $userNameClient, *objPath, *dataId);
}
dataObjRead(*dataObj, *len, *buf) {
    (*desc, *objPath, *dataId) = *dataObj;
    msiDataObjRead(*desc, *len, *buf);
    sendAccessDataObj(ACCESS_TYPE_DATA_OBJ_READ, $userNameClient, *objPath, *dataId);
}
dataObjWrite(*dataObj, *buf, *len) {
    (*desc, *objPath, *dataId) = *dataObj;
    msiDataObjWrite(*desc, *buf, *len);
    sendAccessDataObj(ACCESS_TYPE_DATA_OBJ_WRITE, $userNameClient, *objPath, *dataId);
}
dataObjClose(*dataObj, *status) {
    (*desc, *objPath, *dataId) = *dataObj;
    msiDataObjClose(*desc, *status);
    # the close action has triggered peps, don’t send access here
}
dataObjUnlink(*objPath, *options, *status) {
    if (*options == "") {
        *kvpStr = *objPath;
    } else {
        *kvpStr = "objPath=*objPath++++*options";
    }
    *dataId = getDataObjId(*objPath);
msiDataObjUnlink(*kvpStr, *status);
# peps have already been triggered by msiDataObjUnlink
# sendAccessDataObjCurrentUser(ACCESS_TYPE_DATA_OBJ_DELETE, *objPath, *dataId);
}

sendActionCurrentUser(*AccessType, *ActionId, *TimeStart) {
    if(*AccessType == ACCESS_TYPE_RULE) {
        *description = "<img src='images/icons/rule.png'/>*ActionId;  
    } else if(*AccessType == ACCESS_TYPE_MICROSERVICE) {
        *description = "<img src='images/icons/microservice.png'/>*ActionId;  
    }
    sendAction(*AccessType, $userNameClient, *ActionId, *TimeStart, timeStrNow(), *description)
}

actionStart(*actionId) = (*actionId, timeStrNow())

ruleStart(*ruleId) = actionStart(*ruleId)

ruleFinish(*rule) {
    let (*actionId, *timeStart) = *rule in
    sendActionCurrentUser(ACCESS_TYPE_RULE, *actionId, *timeStart);
}

ruleError(*rule) = ruleFinish(*rule)

microserviceStart(*microserviceId) = actionStart(*microserviceId)

microserviceFinish(*rule) {
    let (*actionId, *timeStart) = *rule in
    sendActionCurrentUser(ACCESS_TYPE_MICROSERVICE, *actionId, *timeStart);
}

microserviceError(*rule) = microserviceFinish(*rule)

replace(*str, *src, *tgt) {
    if(*src == "") {
        *str;
    } else {
        *strlen = strlen(*str);
        *srclen = strlen(*src);
        *newstr = "";
        *cp = 0;
        for(*i=0; *i<*strlen-*srclen+1; *i=*i+1) {
            if(substr(*str, *i, *i+*srclen) == *src) {
                *newstr = *newstr ++ substr(*str, *cp, *i) ++ *tgt;
                *i = *i+srclen-1;
                *cp = *i+1;
            }
        }
        *newstr = *newstr ++ substr(*str, *cp, *strlen);
        *newstr;
    }
}

strstr(*str, *src) {
    *strlen = strlen(*str);
    *srclen = strlen(*src);
    *i = 1;
    for(*i=0; *i<*strlen-*srclen+1; *i=*i+1) {
        if(substr(*str, *i, *i+*srclen) == *src) {
            *i = *i+srclen;
            break;
        }
    }
    *i;
}
6 Protected Data Policy Sets

The UNC requirements for management of protected data sets have been analyzed for development of computer actionable policies that can automate management tasks.

The data management requirements are abstracted from the document, https://www.med.unc.edu/security/hipaa/documents/ADMIN0082%20Info%20Security.pdf. The requirements are listed in Appendix E. Each requirement has been evaluated for the feasibility of creating a computer actionable policy that automates enforcement. Policies are also defined to verify that each requirement have been enforced.

A deep archive is proposed for managing data that contains “Protected” information at UNC. No access is permitted from the external world to the deep archive. Instead processes running within the deep archive pull data records from a staging area. On the staging area, the data sets are checked for “Protected” information, encrypted, and stored into the deep archive, as shown in Figure 1.

![UNC HIPAA Archive](image)

Figure 1. Federated data grids for a deep archive

The “Protected” records may also be archived at an “off-site” location such as the Texas Advanced Computer Center to minimize risk of data loss. The iRODS data grid authenticates every user, authorizes every operation, manages interactions with the storage systems, and creates an event database detailing every interaction. Policies
can parse the event database to verify compliance with policies over time, track unauthorized access attempts, and track data corruption events. Group permissions are defined for access to the data to simplify user management.

The tasks for protected data are listed in Table 2.

Table 2. Protected data tasks requiring policy control

1. Check for presence of PII on ingestion
2. Check for viruses on ingestion
3. Check passwords for required attributes
4. Encrypt data on ingestion
5. Encrypt data transfers
6. Federation - control data copies (access control)
7. Federation - manage remote data grid interactions (update rule base)
8. Federation - periodically copy data
9. Federation - manage data retrieval (update access controls)
10. Generate checksum on ingestion
11. Generate report of corrections to data sets or access controls
12. Generate report for cost (time) required to audit events
13. Generate report of types of protected assets present within a collection
14. Generate report of all security and corruption events
15. Generate report of the policies that are applied to the collections
16. List all storage systems being used
17. List persons who can access a collection
18. List staff by position and required training courses
19. List versions of technology that are being used
20. Maintain document on independent assessment of software
21. Maintain log of all software changes, OS upgrades
22. Maintain log of disclosures
23. Maintain password history on user name
24. Parse event trail for all accessed systems
25. Parse event trail for all persons accessing collection
26. Parse event trail for all unsuccessful attempts to access data
27. Parse event trail for changes to policies
28. Parse event trail for inactivity
29. Parse event trail for updates to rule bases
30. Parse event trail to correlate data accesses with client actions
31. Provide test environment to verify policies on new systems
Table 2 continued. Protected data tasks requiring policy control

<table>
<thead>
<tr>
<th>Task Number</th>
<th>Task Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Provide test system for evaluating a recovery procedure</td>
</tr>
<tr>
<td>33</td>
<td>Provide training courses for users</td>
</tr>
<tr>
<td>34</td>
<td>Replicate data sets on ingestion</td>
</tr>
<tr>
<td>35</td>
<td>Replicate iCAT periodically</td>
</tr>
<tr>
<td>36</td>
<td>Set access approval flag</td>
</tr>
<tr>
<td>37</td>
<td>Set access controls</td>
</tr>
<tr>
<td>38</td>
<td>Set access restriction until approval flag is set</td>
</tr>
<tr>
<td>39</td>
<td>Set approval flag per collection for enabling bulk download</td>
</tr>
<tr>
<td>40</td>
<td>Set asset protection classifier for data sets based on type of PII</td>
</tr>
<tr>
<td>41</td>
<td>Set flag for whether tickets can be used on files in a collection</td>
</tr>
<tr>
<td>42</td>
<td>Set lockout flag and period on user name - counting number of tries</td>
</tr>
<tr>
<td>43</td>
<td>Set password update flag on user name</td>
</tr>
<tr>
<td>44</td>
<td>Set retention period for data reviews</td>
</tr>
<tr>
<td>45</td>
<td>Set retention period on ingestion</td>
</tr>
<tr>
<td>46</td>
<td>Track systems by type (server, laptop, router,....)</td>
</tr>
<tr>
<td>47</td>
<td>Verify approval flags within a collection</td>
</tr>
<tr>
<td>48</td>
<td>Verify files have not been corrupted</td>
</tr>
<tr>
<td>49</td>
<td>Verify presence of required replicas</td>
</tr>
<tr>
<td>50</td>
<td>Verify that no controlled data collections have public or anonymous access</td>
</tr>
<tr>
<td>51</td>
<td>Verify that protected assets have been encrypted</td>
</tr>
</tbody>
</table>

For each listed task, we demonstrate an iRODS policy that implements the associated data management functions.

6.1 Check for presence of PII on ingestion (Policy 34)
The bitcurator technology is able to parse binary images for personally identified information such as credit card numbers and social security numbers. The current implementation runs the bitcurator executable on the storage system holding the data. The bitcurator technology is described in section 5.8.

6.2 Check for viruses on ingestion (Policy 31)
All files in a staging area can be checked for the presence of a virus. When the check is complete, the files can then be moved into a collection. This uses the clamscan virus check routine which is run as an external executable. The clamscan program must be installed on the iRODS server where the staging area is located in the /usr/bin directory.
6.2.1 Scan files and flag infected objects

The rule for invoking virus detection are listed in Section 5.14.1. The rule runs the clamscan script on an external resource, which checks for the presence of viruses. Each file is flagged with a metadata attribute to record the status of the virus check.

6.2.2 Migrate files that pass the virus check

A query can be made to the catalog to identify files that have passed the virus check. The good files are migrated to the archive, and the virus flag is reset.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-migrate-files.r.

```r
migrateFiles {
  # hipaa-migrate-files.r
  # Migrate files that have the attribute VIRUS_SCAN_PASSED
  *Q1 = select DATA_NAME, COLL_NAME where META_DATA_ATTR_NAME = "VIRUS_SCAN_PASSED";
  foreach (*R1 in *Q1) {
    *File = *R1.DATA_NAME;
    *Coll = *R1.COLL_NAME;
    *Path = *Coll/*File;
    *Q2 = select META_DATA_ATTR_VALUE where DATA_NAME = "*File" and COLL_NAME = "*Coll" and META_DATA_ATTR_NAME = "VIRUS_SCAN_PASSED";
    foreach (*R2 in *Q2) {
      *Val = *R2.META_DATA_ATTR_VALUE;
      # Remove the metadata attribute
      *Str1 = "VIRUS_SCAN_PASSED=*Val";
      msiString2KeyValPair(*Str1, *kvp1);
      msiRemoveKeyValuePairsFromObj(*kvp1, *Path, "-d");
      # Insert a revised attribute
      *Str2 = "VIRUS_SCAN_PASS=*Val";
      msiString2KeyValPair(*Str2, *kvp2);
      msiAssociateKeyValuePairsToObj(*kvp2, *Path, "-d");
      # Move the file to the archive
    }
    *Dest = "/UNC-HIPAA/home/HIPAA/Archive/" ++ *File;
    msiDataObjRename(*Path, *Dest, "0", *Status);
  }
}
```

6.3 Check passwords for required attributes (Policy 35)

The policy enforcement point acCheckPasswordStrength checks password strength (added after iRODS 3.2), and is called when the admin or user sets a password. By default, this is a no-op but the simple rule example below can be used to enforce a minimal password length. The password may also require at least one number. This check may be done by an external authentication manager instead of within iRODS.

Updates to this policy are available from
6.4 Encrypt data on ingestion (Policy 36)

The iRODS data grid supports SSL encryption on data transfers. The same encryption can be accessed through a micro-service to encrypt data on storage. The example rule automates encryption on files submitted to the collection:

/UNC-CH/home/HIPAA/Archive

The goal is to maintain data as an encrypted file during transport, as well as within storage.

The rule is implemented as a policy that is enforced at the acPostProcForPut policy enforcement point. A flag is set on the file to denote that encryption has been done. The metadata attribute DATA_ENCRYPT value is set to 1. The rule is located in the acPostProcForPut-encrypt.re file.

Updates to this policy are available from
6.5 Encrypt data transfers (Policy 37)

The iRODS data grid can be set up to use SSL, and automatically encrypt data transfers. This is a configuration setting that is controlled by environment variables:

- **irodsSSLCertificateChainFile (server)** - the file containing the server's certificate chain. The certificates must be in PEM format and must be sorted starting with the subject's certificate (actual client or server certificate), followed by intermediate CA certificates if applicable, and ending at the highest level (root) CA.

- **irodsSSLCertificateKeyFile (server)** - private key corresponding to the server's certificate in the certificate chain file.

- **irodsSLLDHPramsFile (server)** - the Diffie-Hellman parameter file location.

- **irodsSSLVerifyServer (client)** - what level of server certificate based authentication to perform. 'none' means not to perform any authentication at all. 'cert' means to verify the certificate validity (i.e. that it was signed by a trusted CA). 'hostname' means to validate the certificate and to verify that the irodsHost's FQDN matches either the common name or one of the subjectAltNames of the certificate. 'hostname' is the default setting.

- **irodsSSLCACertificateFile (client)** - location of a file of trusted CA certificates in PEM format. Note that the certificates in this file are used in conjunction with the system default trusted certificates.

- **irodsSSLCACertificatePath (client)** - location of a directory containing CA certificates in PEM format. The files each contain one CA certificate. The files are looked up by the CA subject name hash value, which must hence be available. If more than one CA certificate with the same name hash value exist, the extension must be different (e.g. 9d66eef0.0, 9d66eef0.1 etc). The search is performed in the ordering of the extension number, regardless of other properties of the certificates. Use the 'c_rehash' utility to create the necessary links.

6.6 Federation - control data copies (Policy 38)

A primary concern is that protected files in a federation retain appropriate access controls. One way to achieve this is to copy the metadata attributes for each file along with the data, and then run the same ACCESS_APPROVAL policies in the federated data grid.

This rule copies access controls and metadata attributes for a file. This assumes that equivalent accounts exist in both data grids. This requires upgrades to support a federated data grid for msiCopyAVUMetadata and msiLoadACLFromDataObj. The rule uses the policy functions:

- `checkCollInput`
- `isData`

Updates to this policy are available from
CopyACLAVU {
    # hipaa-acl-AVU-copy.r
    #Rule to copy access controls and AVUs to a file in a collection
    #Input
    # Collection that will be used as source
    # Zone that will receive copies of AVUs and ACLs
    #Policy assumes that the account names are the same for both src and dest
    #Output
    # List of files that were updated
    # Generate home collection name for user running the rule
    *Collsrc = "/$rodsZoneClient/home/$userNameClient/" ++ *Coll;
    *Colldest = "/*Zone/home/$userNameClient/" ++ "#" ++ "$rodsZoneClient/" ++ "/" ++ *Coll;
    
    #Verify input path is a collection
    checkCollInput (*Collsrc);
    
    #Verify output path is a collection
    checkCollInput (*Colldest);
    
    *Query3 = select DATA_NAME, DATA_ID where COLL_NAME = '*Collsrc';
    foreach(*Row3 in *Query3) {
        *File = *Row3.DATA_NAME;
        *Sdataid = *Row3.DATA_ID;
        *Pathsrc = *Collsrc ++ "/" ++ *File;
        *Pathdest = *Colldest ++ "/" ++ *File;
        # verify destination file exists
        isData (*Colldest, *File, *Status);
        *Status1 == "0";
        if (*Status == "0") {
            writeLine("stdout","Destination file " ++ *Pathdest ++ " does not exist");
            msiDataObjCopy(*Pathsrc, *Pathdest, "verifyChksum=", *Status1);
        } if (*Status1 == "0") {
            msiSetACL("default", "own", "$userNameClient#$rodsZoneClient", *Dest1);
            *Query4 = select DATA_ACCESS_TYPE, DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = *Sdataid;
            #Loop over access controls for the file
            foreach(*Row4 in *Query4) {
                *Userdid = *Row4.DATA_ACCESS_USER_ID;
                *Datatype = *Row4.DATA_ACCESS_TYPE;
                if(*Userid != *Userdid) {
                    *Query5 = select TOKEN_NAME where TOKEN_NAMESPACE = 'access_type' and TOKEN_ID = *Datatype;
                    foreach(*Row5 in *Query5) {*Access = *Row5.TOKEN_NAME;}
                    *Query6 = select USER_NAME, USER_ZONE where USER_ID = *Userdid;
                    foreach(*Row6 in *Query6) {
                        *Usern = *Row6.USER_NAME;
                        *Userz = *Row6.USER_ZONE;
                    }
                    msiSetACL("default", *Access, "$Usern#$Userz", *Dest1);
                    writeLine("stdout","Path has access control " ++ *Access ++ " for user " ++ *Usern);
                }
            }
            # copy AVUs
            *Q2 = select META_DATA_ATTR_NAME, META_DATA_ATTR_VALUE, META_DATA_ATTR_UNITS where DATA_ID = *Sdataid;
        }
    }
}
foreach (*R2 in *Q2) {
    *Attn = *R2.META_DATA_ATTR_NAME;
    *Attv = *R2.META_DATA_ATTR_VALUE;
    *Attu = *R2.META_DATA_ATTR_UNITS;
    # This policy uses a micro-service developed by the iPlant Collaborative
    msiSetAVU ("-d", *Dest1, *Attn, *Attv, *Attu);
}
writeLine("*file", "Moved file *Src1 to *Dest1");
# Delete file from staging area
    msiDataObjUnlink("objPath=*Src1+++forceFlag=", *Status);
}
}
INPUT *Coll = "$sub1", *Zone = "tempZone"
OUTPUT ruleExecOut

6.7 Federation - manage remote data grid interactions (Policy 32)
When two data grids are federated, decisions have to be made about compatibility of the data management policies. If the desire is to have both data grids implement the same policies, then the policies from the UNC grid will need to be loaded into the federated data grid. This is of particular importance for ensuring:

- Access controls
- Retention flags
- Protected information
- Encryption
- Approval flags

6.7.1 Updating rule base across servers
The rule engine in iRODS reads a local copy of the rule base to improve performance. Coordination of the multiple rule bases is needed when policies are updated. This rule set, developed by Chris Smith, stores the rules in the iCAT metadata catalog, extracts rules from the catalog into a file, and then updates each of the server rule bases.

6.7.1.1 Storing rules in the DB from a source file.
This rule is run on the master ICAT.

Updates to this policy are available from

    idsStoreRules {
    # hipaa-idsStore.r
    msiAdmReadRulesFromFileIntoStruct(*inFileName, *struct);
    msiAdmInsertRulesFromStructIntoDB(*ruleBase, *struct);
    }
INPUT *ruleBase="IDSbase", *inFileName="ids-src"
OUTPUT ruleExecOut
6.7.1.2 **Prime the ICAT's rule base**

This rule is run on the master catalog.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-idsApply.r.

```r
idsApplyRules {
  # hipaa-idsApply.r
  if (*rloc == "localhost") {
    msiAdmRetrieveRulesFromDBIntoStruct(*ruleBase, "0", *struct);
    msiAdmWriteRulesFromStructIntoFile(*outFileName, *struct);
  }
  else {
    remote(*rloc, "null") {
      msiAdmRetrieveRulesFromDBIntoStruct(*ruleBase, "0", *struct);
      msiAdmWriteRulesFromStructIntoFile(*outFileName, *struct);
    }
  }
}

INPUT *ruleBase="IDSbase", *outFileName="ids", *rloc="localhost"
OUTPUT ruleExecOut
```

6.7.1.3 **Push rules to resource servers**

This rule pushes the rules to all the resource servers. For servers that don't host resources, a separate rule will need to be run at each server to prime the local rule base from the iCAT catalog.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-idsPush.r.

```r
idsPushRules {
  # hipaa-idsPush.r
  *ContInxOld = 1;
  msiMakeGenQuery("RESC_LOC", "RESC_LOC != 'localhost'", *GenQInp);
  msiExecGenQuery(*GenQInp, *GenQOut);
  msiGetContInxFromGenQueryOut(*GenQOut, *ContInxNew);
  while (*ContInxOld > 0) {
    foreach (*GenQOut) {
      msiGetValByKey(*GenQOut, "RESC_LOC", *rloc);
      writeLine("stdout", "Pushing rule base *ruleBase to *rloc");
      remote(*rloc, "null") {
        msiAdmRetrieveRulesFromDBIntoStruct(*ruleBase, "0", *struct);
        msiAdmWriteRulesFromStructIntoFile(*outFileName, *struct);
      }
    }
    *ContInxOld = *ContInxNew;
    if (*ContInxOld > 0) {
      msiGetMoreRows(*GenQInp, *GenQOut, *ContInxNew);
    }
  }
}

INPUT *ruleBase="IDSbase", *outFileName="ids"
```
A second approach is to allow the federated data grid to implement a separate set of policies, but restrict file exchange between the data grids to data that does not require protection. This can be controlled by forcing all data exchanges to be done with data that have anonymous access.

This restriction is implemented by not allowing any member of the federated data grid to have an account in the UNC data grid. This minimizes the opportunity to give inappropriate access to data within the UNC data grid.

### 6.8 Federation – Copy Data from staging area (Policy 20)

Files can be staged between two data grids. This rule recursively copies files from a staging area into a second data grid, checks that the files do not already exist in the second data grid, verifies checksums after the copy, and sets access permissions. The rule uses the policy functions:

- `checkCollInput`
- `checkRescInput`
- `createLogFile`
- `findZoneHostName`
- `isColl`

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-stageFederation.r.

```r
myStagingRule {
    # hipaa-stageFederation.r
    # Loop over files in a collection, *Src
    # Put all files into a staging collection, *Dest
    checkCollInput (*Src);
    checkCollInput (*Dest);
    checkRescInput (*Res, *DestZone);
    *Len = strlen(*Src);

    #=============get current time, Timestamp is YYY-MM-DD.hh:mm:ss ==============
    msiGetSystemTime("TimeA,"unix");

    #============== create a collection for log files if it does not exist ==============

    #============== find files to stage
    *Query = select DATA_NAME, DATA_CHECKSUM, COLL_NAME, DATA_MODIFY_TIME where COLL_NAME like "*Src%";
    foreach(*Row in *Query) {
        *File = *Row.DATA_NAME;
        *Check = *Row.DATA_CHECKSUM;
        *Coll = *Row.COLL_NAME;
        if(*Coll != *LPath) {
            *L1 = strlen(*Coll);
```
*Src1 = *Coll ++ "/" ++ *File;
*C1 = substr(*Coll,*Len,*L1);
if(strlen(*C1)==0) {
    *DestColl = *Dest;
    *Dest1 = *Dest ++ "/" ++ *File;
} else {
    *DestColl = *Dest ++ *C1;
    *Dest1 = *Dest ++ *C1 ++ "/" ++ *File;
}
isColl(*DestColl,*Lfile,*Status);
if (*Status >= 0) {
    msiDataObjCopy(*Src1,*Dest1,"destRescName=*Res++++forceFlag=",*Status);
    msiSetACL("default","own",*Owner,*Dest1);
    msiDataObjChksum(*Dest1,"forceChksum=",*Chksum);
    if (*Check != *Chksum) {
        writeLine("*Lfile","Bad checksum for file *Dest1");
    }
    else {
        writeLine("*Lfile","*Src1 copied to *Dest1 *Check *TimeH");
    }
}
}

INPUT *Res=$"demoResc", *DestZone =$"tempZone",
*Src=$"/rodsZoneClient/home/$userNameClient/sub1",
*Dest=$"/*DestZone/home/*Owner/sub2", *Owner = "odum_fed#dfcmain"
OUTPUT ruleExecOut

6.9 Federation- manage data retrieval (Policy 39)
Inappropriate data retrieval can be controlled from a federation by applying
the same access controls and policies across the federated data grid. This is necessary
because the federated data grid can be accessed directly, independently of the
original data grid.

If access is done through the original data grid, accounts can be established in the
federated data grid to control data retrieval. The accounts reference the original
data grid:

- Account name: UNC-HIPAA#HIPAA
- Account name: UNC-HIPAA#public
- Account name: UNC-HIPAA#gridAdmin

Access controls can then be applied in the federated data grid for each account in
the original data grid.

This rule generates a pipe-delimited file of user accounts in the data grid. The rule
uses the policy functions:
- checkRescInput
- createLogFile
- findZoneHostName
- isColl
**createAccounts**

```r
createAccounts {
# hipaa-create-accounts.r
# Creates a time-stamped pipe-delimited file for the accounts in a data grid
checkRescInput (*Res, $rodsZoneClient);
*Coll = "/$rodsZoneClient/home/$userNameClient/"
*Q1 = select USER_NAME, USER_TYPE;
*Zone = $rodsZoneClient;
foreach (*R1 in *Q1) {
*User = *R1.USER_NAME;
*Type = *R1.USER_TYPE;
writeLine("*Lfile", "*User|001|*Type|*Zone|"
}
msiDataObjClose(*L_FD,*Status);
}
```

**Input**
*Res = "uncResc", *Accounts = "Account"

**Output**
ruleExecOut

This rule reads an Account file to generate new accounts. Note that the account file needs to be copied into the federated data grid. The command must also be run in the federated data grid. The account names are created in the form

User_name#zone_name

Note that the micro-service msiCreateUserAccountsFromDataObj is used to load the accounts. This micro-service is not yet ported to iRODS version 4.2. The rule uses the policy function:

checkPathInput

**myTestRule**

```r
myTestRule {
# hipaa-accountImport.r
# Input parameters are:
# Path
# File format for user accounts is
# User-name|User-ID|User-type|Zone|
# guest001|rodsuser|tempZone
# Output parameter is:
# Status
checkPathInput (*Path);
msiCreateUserAccountsFromDataObj(*Path,*Status);
writeLine("stdout", "Add user accounts defined in file *Path");
}
```

**Input**
*Path="/rsZoneClient/home/$userNameClient/Accounts/Account-2015-08-27.22:12:56"

**Output**
ruleExecOut

Updates to this policy are available from
6.10 Generate checksum on ingestion (Policy 40)
A checksum is generated for every file that is put into the data grid. This rule is located in the acPostProcForPut-checksum.re file.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acPostProcForPut-checksum.re.

    acPostProcForPut {msiSysChksumDataObj; }

6.11 Generate report of corrections to data sets or access controls (Policy 41)
The audit log can be parsed to identify all changes to data sets or access controls. We assume that any file for which a new version has been created constitutes a correction to a data set.

The auditing capability depends on a set of external services and rules. The following services are used: ElasticSearch, OSGi, and AMQP. ServiceMix provides both OSGi and AMQP. On the iRODS server, auditing requires a list of iRODS rules, and client libraries for sending messages to the AMQP service. In addition, networking on the servers running these services must be configured to allow these services to communicate.

The rules that need to be installed include: databook_pep.re, databook.re, and amqp.re. The rule set databook_pep.re overrides the default iRODS PEPs so that messages are sent for auditing. This has the limitation that if you already have customized PEPs, you have to manually edit them. Alternatively, starting from iRODS 4.2, you can install the auditing plugin which will allow you to avoid changing your customized PEPs. The rule set databook.re provides the main functionality for auditing. The rule set amqp.re provides rules for interacting with AMQP. In addition, Python libraries are used to send messages to AMQP. These can be set up using an automated setup script from the source repository, although customizing the script is usually necessary in order to achieve a particular set up.

Once the auditing services are installed, all system access information is stored in an Elasticsearch index. The index can be queried. An administrator can retrieve events based on the following parameters:

- fromDate: from which date
- toDate: to which date
- event: the event
- pid: uri filter
- start: starting index
- and count: how many results to return

A Java program is used to interact with Elasticsearch. This example generates the number of access events per file for reporting to DataONE. The results can be limited to a date range. The EventsEnum defines which type of event to monitor.
The types of events that are monitored are listed in org.dataone.service.types.v1.Event:

- put
- data object put
- get
- data object get
- overwrite
- data object overwrite
- delete
- data object delete
- replicate
- data object replicate
- synch_failure
- data object synch_failure

The Java is available at https://github.com/DICE-UNC/policy-workbook/blob/master/dfc-elasticsearch.java

```java
// use Elastic search query to retrieve list of logEntry (events)
//log?[fromDate={fromDate}][&toDate={toDate}][&event={event}][&pidFilter={pidFilter}][&start={start}][&count={count}]
@override
public Log getLogs(
    Date fromDate,
    Date toDate,
    EventsEnum event,
    String pidFilter,
    int startIdx,
    int count)
    throws NoNodeAvailableException {

    boolean datesExist = true;
    if (fromDate == null && toDate == null) {
        datesExist = false;
    }

    // restrict search to DataONE exposed data objects
    // get this list from the Handle server
    // should return something like this:
    // "/dfc3/home/rods/fabfile.py2".*|"/dfc3/home/rods/fabfile.py3".*
    String uriRegex = null;
    try {
        uriRegex = getDataOneDataObjectsRegex();
    } catch (JargonException e) {
        logger.error("error getting DataOne uids and converting to regex");
        throw new NoNodeAvailableException(e.getMessage());
    }

    // get elasticsearch properties
    PropertiesLoader loader = new PropertiesLoader();
    String elasticsearchDNS = loader.getProperty("irods.dataone.events.elasticsearch.dns");
    int elasticsearchPort = Integer.parseInt(loader.getProperty("irods.dataone.events.elasticsearch.port"));
    String searchIndex = loader.getProperty("irods.dataone.events.elasticsearch.searchindex");
    String searchType = loader.getProperty("irods.dataone.events.elasticsearch.searchtype");
    String clusterName =
```
loader.getProperty("irods.dataone.events.elasticsearch.cluster.name");
String rangeField = "created";

BoolQueryBuilder boolQuery = QueryBuilders.boolQuery()
  .must(QueryBuilders.matchQuery("type", "databook.persistence.rule.rdf.ruleset.Access"));
if (event != null) {
  boolQuery.must(QueryBuilders.matchQuery("title", event.getDatabookEvent()));
}
if (uriRegex != null && !uriRegex.isEmpty()) {
  boolQuery.must(QueryBuilders.regexpQuery("url", uriRegex));
}

// Note that date time precision is limited to one millisecond.
RangeFilterBuilder filterBuilder = FilterBuilders.rangeFilter(rangeField);
if (datesExist) {
  if (fromDate != null) {
    filterBuilder
      .from(fromDate.getTime())
      .includeLower(true);
  }
  else {
    filterBuilder
      .from(0)
      .includeLower(true);
  }
  if (toDate != null) {
    filterBuilder
      .to(toDate.getTime())
      .includeUpper(false);
  }
  else {
    filterBuilder
      .to(System.currentTimeMillis())
      .includeUpper(false);
  }
}

logger.info("creating elastic search transport client: dns={}, port={}", elasticsearchDNS,
elasticsearchport);
Settings settings = ImmutableSettings.settingsBuilder()
  .put("cluster.name", clusterName).build();
Client client = null;
if(elasticsearchDNS != null && elasticsearchDNS.length() > 0) {
  client = new TransportClient(settings)
    .addTransportAddress(new InetSocketTransportAddress(elasticsearchDNS,
elasticsearchport));
}
else {
  client = new TransportClient(settings)
    .addTransportAddress(new InetSocketTransportAddress("localhost", 9300));
}

SearchRequestBuilder srBuilder = client.prepareSearch(searchIndex)
  .setTypes(searchType)
  .setQuery(boolQuery)
  .setFrom(startIdx).setSize(startIdx + count);
if (datesExist) {
  srBuilder.setPostFilter(filterBuilder);
}
logger.info("getLogs: built search request: {}", srBuilder.toString());
SearchResponse response = srBuilder.execute().actionGet();
logger.info("getLogs: got search response: {}", response.toString());
SearchHit[] searchHits = response.getHits().getHits();
Log log = new Log();
if (searchHits.length > 0) {
    // now put retrieved data into Log object
    try {
        log = copyHitsToLog(startIdx, count, response.getHits().getTotalHits(),
                            searchHits);
    } catch (JargonException e) {
        logger.error("error copying elastic search hits into Log entries");
        throw new NoNodeAvailableException(e.getMessage());
    }
}
return log;

6.12 Generate report for cost (time) required to audit events (Policy 42)
This rule queries the event index to identify the amount of time needed to run an audit. The execution time of the Java script for accessing ElasticSearch is saved to create the cost report.

6.13 Generate report of types of protected assets (Policy 43)
A summary report can be generated that counts the number of files within a collection for each type of asset classifier:
- 1- Protected Health Information – PHI
- 2 - Personally Identifiable Information – PII such as social security numbers
- 3 - Payment Card Information – PCI such as account numbers, card holder name, expiration date, service code, CID, PINs
- 4 - Legally restricted data – classified
- 5 - Proprietary information
The rule uses the policy function:
checkCollInput

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-asset-report.r.

assetClassifier {
    # hipaa-asset-report.r
    # Count the number of files with each type of classifier in a collection
    checkCollInput(*Coll);
    *Q1 = select count(DATA_ID) where COLL_NAME = *Coll and META_DATA_ATTR_NAME = "AssetProtectionClassifier" and META_DATA_ATTR_VALUE = "1";
    foreach(*R1 in *Q1) {*C1 = *R1.DATA_ID;}
    *Q2 = select count(DATA_ID) where COLL_NAME = *Coll and META_DATA_ATTR_NAME = "AssetProtectionClassifier" and META_DATA_ATTR_VALUE = "2";
    foreach(*R2 in *Q2) {*C2 = *R2.DATA_ID;}
    *Q3 = select count(DATA_ID) where COLL_NAME = *Coll and META_DATA_ATTR_NAME = "AssetProtectionClassifier" and META_DATA_ATTR_VALUE = "3";
    foreach(*R3 in *Q3) {*C3 = *R3.DATA_ID;}
    *Q4 = select count(DATA_ID) where COLL_NAME = *Coll and META_DATA_ATTR_NAME = "AssetProtectionClassifier" and META_DATA_ATTR_VALUE = "4";
    foreach(*R4 in *Q4) {*C4 = *R4.DATA_ID;}
}
*Q5 = select count(DATA_ID) where COLL_NAME = *Coll and META_DATA_ATTR_NAME = "AssetProtectionClassifier" and META_DATA_ATTR_VALUE = "5";
foreach(*R5 in *Q5) {*C5 = *R5.DATA_ID;}
writeLine("stdout", "Number of PHI files is *C1");
writeLine("stdout", "Number of PII files is *C2");
writeLine("stdout", "Number of PCI files is *C3");
writeLine("stdout", "Number of classified files is *C4");
writeLine("stdout", "Number of proprietary files in *C5");
}
INPUT *Coll="/UNC-HIPAA/home/HIPAA"
OUTPUT ruleExecOut

6.14 Generate report of all security and corruption events (Policy 44)
The audit log can be parsed to identify all access events, and correlate the access
with an authentication event. If an access event cannot be correlated to an
authentication event, a possible security event can be logged.

For corruption events, use the policy in Section 14. This identifies and lists all files
that have been corrupted.

6.15 Generate report of the policies applied to collections (Policy 45)
Within the iRODS data grid, policies are stored in the iCAT metadata catalog. The
policies are versioned, such that each policy change creates a new version. The
policies can be extracted from the catalog, distributed to each site where data are
stored, and instantiated as a distributed rule base that controls operations within
the data grid.

The iRODS data grid relies upon a distributed rule engine and distributed rule bases
to implement policies. If a policy is changed, for consistency the revised rule base
needs to be installed at each server location.

6.15.1 Deploy rule sets
This rule identifies the servers, and uploads a new version of the rule base to each
server. The micro-services used by this rule are available at
https://github.com/DICE-UNC/irods_rule_admin_microservices

Updates to this policy are available from

copyRule {
  # hipaa-deploy-rules.r
  # the copy.re must be available on all servers
  # the rule admin microservices must also be available on all servers
  remoteDeployRuleSets(
    *ruleBaseName, 
    *targets
  );
}
remoteDeployRuleSets(*rbs, *addrs) {
  *out = "";
}
foreach(*addr in *addrs) {
    foreach(*rb in *rbs) {
        *err = errorCode(remoteWriteRuleSet(*rb, *addr));
        *out = *out ++ "*rb -> *addr " ++ (if *err != 0 then "failure" else "success") ++ \\
            "\n";
    }
}
writeLine("stdout", *out);
}

remoteWriteRuleSet(*rb, *addr) {
    msiReadRuleSet(*rb, *rule);
    msiChksumRuleSet(*rb, *chksum);
    remote(*addr, "") {
        writeRuleSet(*rb, *rule, *chksum);
    }
}

INPUT *ruleBaseName=list("core"),*targets=list("localhost")
OUTPUT ruleExecOut

6.15.2 Update rule sets
This policy function reads and writes rule sets that have been deposited into the icAT catalog.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-write-rules.r.

writeRuleSet(*rb, *rule, *chksum) {
    # hipaa-write-rules.r
    backupRuleSet(*rb, *rbak):::
    if(errorcode(*rbak) == 0) {
        msiMvRuleSet(*rbak, *rb);
    }
};
msiWriteRuleSet(*rb, *rule):::
    *ec = errorCode(msiRuleSetExists(*rb, *e));
    if(*ec == 0) {
        if(*e == 0) {
            msiRmRuleSet(*rb);
        }
    }
};
msiChksumRuleSet(*rb, *chksum2);
if(*chksum != *chksum2) {
    failmsg(-1, "chksum failed");
}
}

This policy function creates a rule set backup.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-backup-rules.r.
backupRuleSet(*rb, *rbak) {
    # hipaa-backup-rules.r
    *i = 0;
    while(true) {
        *rbak = "*rb.bak*i"
        *ec = errorcode(msiRuleSetExists(*rbak, *e));
        if(*ec != 0) {
            break;
        } else if(*e != 0) {
            break;
        }
        *i = *i + 1;
    }
    msiMvRuleSet(*rb, *rbak);
}

6.15.3 Print rule sets
This rule prints the rule set used by iRODS by listing the core.re file.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-print-rules.r.

    printRules {
        # hipaa-print-rules.r
        # print the rules in the core.re file
        msiAdmShowIRB();
    }
    INPUT null
    OUTPUT ruleExecOut

6.16 List all storage systems being used (Policy 46)
This rule lists the storage systems that are attached to the data grid.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-list-storage.r.

    listStorage {
        # hipaa-list-storage.r
        # List all storage systems in the data grid
        *Q1 = select RESC_NAME;
        foreach (*R1 in *Q1) {
            *Resc = *R1.RESC_NAME;
            writeLine("stdout", "Zone $rodsZoneClient has storage resource *Resc");
        }
    }
    INPUT null
    OUTPUT ruleExecOut

6.17 List persons who can access a collection (Policy 47)
For the specified collection, a list is generated of all persons who have access to files in a collection. The rule uses the policy function:
    checkCollInput
Updates to this policy are available from
http://github.com/DICE-UNC/policy-workbook/hipaa-list-access.r.

```
main {
    # hipaa-list-access.r
    # Identify users who have access to a collection
    checkCollInput (*Coll);
    *Q1 = select DATA_NAME, DATA_ID where COLL_NAME = "*Coll";
    foreach (*R1 in *Q1) {
        *File = *R1.DATA_NAME;
        *DataID = *R1.DATA_ID;
        *Q2 = select DATA_ACCESS_USER_ID, DATA_ACCESS_TYPE where DATA_ACCESS_DATA_ID = ''*DataID';
        foreach (*R2 in *Q2) {
            *Userid = *R2.DATA_ACCESS_USER_ID;
            *Type = *R2.DATA_ACCESS_TYPE;
            *Q3 = select USER_NAME where USER_ID = ''*Userid';
            foreach (*R3 in *Q3) {*Name = *R3.USER_NAME;}
            writeLine("stdout","*Name has access to *File in *Coll");
        }
    }
}
```

input *Coll = "/dfcmain/home/rwmoore/test"
output ruleExecOut

6.18 List staff by position and required training courses (Policy 48)
A list of all persons with accounts in the data grid can be generated. The USER_INFO field can be used to annotate the staff position and the last training course through XML tags:

```
USER_INFO = "<Position>staff</Position><Training>course</Training>
```

6.18.1 Set position and training
This policy modifies existing user accounts according to information in an iRODS object. The format of the account file is:

```
user-name|field|new-value
```
where valid fields include:

- type
- zone
- comment
- info
- password

A file containing the desired updates is loaded into the Reports directory. The rule uses the policy function:

```
checkPathInput
```

Updates to this policy are available from

```
myTestRule {
```
# hipaa-update-user-info.r
# Input parameter is:
# Path of file containing information
# Output parameter is:
# Format of the file is
# user-name|field|new-value
# hippaAdmin|info|<Training>Course1</Training>
# Status
checkPathInput(*Path);
msiLoadUserModsFromDataObj(*Path,*Status);
writeLine("stdout","Change info on a user account");

INPUT *Path"/UNC-CH/home/HIPAA/Reports/updateUserInfo"
OUTPUT ruleExecOut

6.18.2 List staff by position and training
A report of all staff positions and the latest training can be generated.

Updates to this policy are available from
http://github.com/DICE-UNC/policy-workbook/hipaa-hipaa-list-training.r.

listTraining {
# hipaa-list-training.r
# List the name, position, and training for each account
*Q1 = select USER_NAME, USER_TYPE, USER_INFO
foreach(*R1 in *Q1) {
 *Name = *R1.USER_NAME;
 *Type = *R1.USER_TYPE;
 *Info = *R1.USER_INFO;
 writeLine("stdout","*Name has role *Type");
 writeLine("stdout"," *Info");
}

INPUT null
OUTPUT ruleExecOut

6.19 List versions of technology that are being used (Policy 49)
A report can be kept in the data grid that identifies the current versions of the
hardware and software technologies used in the preservation environment. This
policy defines the collection location and file name used for the report.

- Technology report name        TechVersionReport
- Collection name               Reports
- Location                      /UNC-CH/home/HIPAA/Reports

Updates to this policy are available from

getTechReport {
# hipaa-tech-report.r
# Gets report from /UNC-CH/home/HIPAA/Reports
# Saves copy on local disk with the name TechVersionReport
*Coll = "/UNC-CH/home/HIPAA/Reports";
In iRODS version 4.x, technologies are plugged into the iRODS framework. By listing all plug-ins, the versions of all hardware and software systems can be automatically tracked. The izonereport command generates a json file that lists the entire iRODS Zone configuration information. The command izonereport validates the information against the schemata found at https://schemas.irods.org.

**6.20 Maintain document on independent assessment of software (Policy 50)**

The report on software assessment can be managed within the data grid. This policy retrieves the specified document from the Report directory.

- Software assessment report name: softwareAssessment
- Collection name: Reports
- Location: /UNC-CH/home/HIPAA/Reports

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-assessment-report.r.

**6.21 Maintain log of all software changes, OS upgrades (Policy 51)**

The log of software changes is maintained by the data grid operators. This policy defines the collection location and file name used for the report.

- Technology report name: LogSoftwareChanges
- Collection name: Reports
- Location: /UNC-CH/home/HIPAA/Reports

A policy to store the log is listed below.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-store-log.r.
6.21.1 Version log files
Each version of a log file can be tracked. When a file is added to the system, a version labeled by the current time stamp is saved, ensuring that a history of changes can be maintained. The version is moved to an archive directory.

The version number can be inserted in the file name before the extension. This rule parses the file name, identifies an extension, and inserts the time stamp before the extension when the version name is created. The ownership of the file is set to the hipaaAdmin account. The rule is listed in section 4.7.1.

6.22 Maintain log of disclosures (Policy 52)
A disclosure log identifies all events associated with unauthorized access to files. The ways this may happen include:

- Incorrect setting of access controls on the files in a collection. One way to detect this is to log all files in a collection that do not have ACCESS_APPROVAL set to 1, but have anonymous or public access.
- Direct reading of the file on disk without going through the data grid. This may happen when a security vulnerability is present within the operating system that has not been patched. Detection of this type of access requires parsing the system log for the computer.
- Unauthorized use of an account. This requires that the unauthorized user learn the password associated with the account. This may happen when a password is shared or stolen. Detection of this type of access requires interaction with the account owner to determine whether they made the access.

In all three cases, a report can be generated that is updated externally to the data grid. The report can be stored in the data grid with versioning enabled, and deletion turned off. The version is stored in Reports/Backup.

This policy defines the collection location and file name used for the report.

- Technology report name: DisclosureReport
- Collection name: Reports
- Location: /UNC-CH/home/HIPAA/Reports
- Version: /UNC-CH/home/HIPAA/Reports/Backup

A rule to store the report uses the policy function:

checkRescInput
findZoneHostName

Updates to this policy are available from
storeSystemLog {
    # hipaa-version-report.r
    # Stores report in /UNC-CH/home/HIPAA/Reports
    # Saves a versioned copy on local disk with the name DisclosureReport
    checkRescInput (*destRescName, $rodsZoneClient);
    *Coll = "/UNC-CH/home/HIPAA/Reports";
    *File = "DisclosureReport";
    *Path = "*Coll/*File";
    # check whether the report already exists
    *Q1 = select count(DATA_ID) where DATA_NAME = '*File' and COLL_NAME = '*Coll';
    foreach (*R1 in *Q1) {*Count = *R1.DATA_ID;}
    if (*Count == '1') {
        # File already exists, create version
        *Dest = '*Coll/Backup/*File';
        msiStoreVersionWithTS(*Path, *Dest, *Status);
    }
    msiDataObjPut(*Path, *destRescName, "localPath=./*File++++forceFlag=", *Status);
}

INPUT *destRescName = "hipaaResc"
OUTPUT ruleExecOut

To turn off deletion on collection /UNC-CH/home/HIPAA/Reports, set the policy enforcement point:

Updates to this policy are available from

acDataDeletePolicy {
    # Rule condition is used to choose which collections to protect
    ON($objPath like "/UNC-CH/home/HIPAA/Reports/*") {
        msiDeleteDisallowed;
    }
}

6.23 Maintain password history on user name (Policy 53)
A history of prior passwords can be kept as events in an external index. The challenge is that the current design does not generate an identity for the user until after the acCheckPasswordStrength has been executed.

One approach is to check the password history after the user name is defined, within the acSetPublicUser Policy enforcement point. Metadata attributes for the prior passwords can then be checked. If a similar prior password is found, a request to change the password can be made and the rule can fail. The metadata attributes are:

- META_USER_ATTR_NAME PasswordHist
- META_USER_ATTR_VALUE prior password
- META_USER_ATTR_UNITS Set to 0 for current password
This policy loads passwords as attributes on the USER_NAME.

Updates to this policy are available from

```re
acSetPublicUserPolicy {
    # reset NumberAttempts and LockoutPeriod
    *User = $userNameClient;
    *Q1 = select META_USER_ATTR_VALUE where USER_NAME = '*User' and
            META_USER_ATTR_NAME = 'NumberAttempts';
    foreach (*R1 in *Q1) {
        *Val = *R1.META_USER_ATTR_VALUE;
        *Str = "NumberAttempts=*Val";
        msiString2KeyValPair(*Str,*Kvp);
        msiRemoveKeyValuePairsFromObj(*Kvp,*User,"-u");
        *Val1 = "0";
        *Str1 = "NumberAttempts=*Val1";
        msiString2KeyValPair(*Str1,*Kvp1);
        msiAssociateKeyValuePairsToObj(*Kvp1,*User,"-u");
    }
    if (int(*Val1) > 5) {
        # set lockout period
        *Q2 = select META_USER_ATTR_VALUE where USER_NAME = '*User' and
            META_USER_ATTR_NAME = 'LockoutPeriod';
        foreach (*R2 in *Q2) {
            *Val = *R2.META_USER_ATTR_VALUE;
            *Str = "LockoutPeriod=*Val";
            msiString2KeyValPair(*Str,*Kvp);
            msiRemoveKeyValuePairsFromObj(*Kvp,*User,"-u");
            *Str1 = "LockoutPeriod=0";
            msiString2KeyValPair(*Str1,*Kvp1);
            msiAssociateKeyValuePairsToObj(*Kvp1,*User,"-u");
        }
    }
    if (*User != 'public' && *User != 'anonymous') {
        *Q2 = select META_USER_ATTR_VALUE where USER_NAME = '*User' and
            META_USER_ATTR_NAME = 'ResetPassword';
        foreach (*R2 in *Q2) {
            *Val = *R2.META_USER_ATTR_VALUE;
            if (*Val == "1") { writeLine("stdout", "Reset your password"); }
        }
    }
    # check prior passwords
    *Q3 = select META_USER_ATTR_VALUE, META_USER_ATTR_UNITS where
        USER_NAME = '*User' and META_USER_ATTR_NAME = 'PasswordHist';
    foreach (*R3 in *Q3) {
        *Pass = *R3.META_USER_ATTR_VALUE;
        *Date = *R3.META_USER_ATTR_UNITS;
        if(*Date == '0') {
```

145
*Passcurrent = *Pass;

}  # do the comparison
foreach (*R3 in *Q3) {
    *Pass = *R3.META_USER_ATTR_VALUE;
    *Date = *R3.META_USER_ATTR_UNITS;
    if (*Date != '0') {
        if (*Pass == *Passcurrent) {
            writeLine("stdout", "Reset your password");
            fail;
        }
    }
}

6.24 Parse event trail for all accessed systems (Policy 54)
The audit log can be queried to identify all accesses to the repository. For each access, the storage resource can be specified. The results can be summarized to identify all of the storage resources that were accessed.

6.25 Parse event trail for all persons accessing collection (Policy 33)
The audit log can be queried to identify all accesses to files in a collection. For each access, the identity of the account making the request is known. The results can be summarized to identify all persons who accessed the collection. See section 5.16.

6.26 Parse event trail for all unsuccessful attempts to access data (Policy 55)
Each access of the data grid is authenticated. If the authentication fails, an event can be generated if the requested operation was a read attempt. The audit log can then be queried to identify all unsuccessful access attempts to files in a collection. The results can be summarized to identify the accounts that had unsuccessful access attempts.

6.27 Parse event trail for changes to policies (Policy 56)
The iRODS data grid can maintain an event database that lists all events associated with managing or accessing the data system. The policies that record events generate messages that are sent to an external indexing system. By searching in the external index, events associated with the policy enforcement points can be identified:

    pep_PLUGINOPERATION_pre
    pep_PLUGINOPERATION_post

Changes to policies should be saved in the iCAT catalog as rule versions using the micro-services

- msiAdmReadRulesFromFileIntoStruct
- msiAdmInsertRulesFromStructIntoDB
The corresponding events in the event database are:

- `pep_msiAdmReadRulesFromFileIntoStruct_pre`
- `pep_msiAdmReadRulesFromFileIntoStruct_post`
- `pep_msiAdmInsertRulesFromStructIntoDB_pre`
- `pep_msiAdmInsertRulesFromStructIntoDB_post`

A query is issued against the event index by issuing a libcurl call:

```
curlGetStr {
    # hipaa-issue-url.r
    # get string from a URL
    msiCurlGetStr(*url, *Buffer);
    writeLine("stdout", str(*Buffer)++" returned string");
}
```

**6.28 Parse event trail for inactivity (Policy 57)**

Each access of the data grid is treated as a separate session. The user is authenticated and the operation is authorized. When the requested operation completes, the session is terminated. Thus users cannot be logged into the data grid without applying operations on the data. Users are only “logged” into the data grid while they are applying operations on their data.

There is the possibility of long-running operations, such as validating checksums for all files in a collection. However, these are expected uses of the system.

**6.29 Parse event trail for updates to rule bases (Policy 58)**

The audit log can be queried to identify all updates made to the policies. Events can be generated that correspond to execution of the micro-service that creates new versions of rules that are registered into the iCAT catalog. The results can be written to a file or printed.

**6.30 Parse event trail to correlate data accesses with client actions (Policy 59)**

Events can be generated for accesses that include the type of client API that was used. Each client API interacts through a plug-in that can track usage events. Events that are tracked include:

- data obj read
- data object update
- data object overwrite
- data object put
- data object get
- data obj read
• data obj write
• data obj create
• data obj remove

6.31 Provide test environment to verify policies on new systems (Policy 60)
The test environment should be an independent iRODS data grid with a separate iCAT catalog, separate storage servers, and disjoint user accounts. The directory structure should be similar to the production environment.

This policy downloads the rules from the test environment, and stores them in a file. We assume the following:
• Test zone is called uncTestZone
• Admin account is called uncTestAdmin
• Test zone rule base is called TestBase
• Rule file is called NewRules

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-export-policies.r.

```r
exportPolicies {
# hipaa-export-policies.r
# Read the rules from the iCAT catalog
# This rule is run on the test system
# Create a file with the rules for import on the production system
  msiGetRulesFromDBIntoStruct (*RuleBase, "0", *Struct);
  msiAdmWriteRulesFromStructIntoFile (*FileName, *Struct);
  msiAdmShowIRB("null");
}
INPUT *RuleBase = "TestBase", *FileName = "NewRules"
OUTPUT ruleExecOut
```

This rule reads the rules from the file NewRules and loads them into the production iCAT catalog.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-import-policies.r.

```r
importPolicies {
# hipaa-import-policies.r
# This rule imports rules from a file into the iCAT catalog
  msiAdmReadRulesFromFileIntoStruct(*FileName,*Struct);
  msiAdmInsertRulesFromStructIntoDB(*RuleBase,*Struct);
  msiAdmShowIRB ("null");
}
INPUT *FileName = "NewRules", *RuleBase = "UNCRules"
OUTPUT ruleExecOut
```
6.32 Provide test system for evaluating a recovery procedure (Policy 61)
A test system would ideally contain a complete set of records from the original data grid, including an up-to-date copy of the metadata catalog. A recovery procedure would then need to do the following steps:

- Recreate the iCAT catalog from the test system. This would set accounts, define storage resources, define file names, define collections
- A checksum on the files would then be run to detect any corrupted files.
- Corrupted files would be replaced from the test system

A replication rule could be run to detect problems. If one of the replicas in the original data grid is still good, this should be sufficient. However, if no good replicas exist, then the file will need to be replaced from the test system. A replication rule is listed in section 4.5.2

6.33 Provide training courses for users (Policy 62)
Information about training courses can be kept in a separate database. For each staff position, a set of required training courses can be defined. The list of required courses can be compared with the courses that were taken, and stored as USER_INFO.

6.34 Replicate data sets on ingestion (Policy 13)
When a file is put into the collection /UNC-CH/home/HIPAA/Archive, it will be replicated to a second storage system. The rule is enforced at the acPostProcForPut policy enforcement point.

Updates to this policy are available from
http://github.com/DICE-UNC/policy-workbook/acPostProcForPut-replicate.re

acPostProcForPut {
  ON($objPath like "/UNC-ARCHIVE/home/Archive/*") {
    delay("<PLUSET>1s</PLUSET>") {
      msiSysReplDataObj('replResc', 'null');
    }
  }
}

6.35 Replicate iCAT periodically (Policy 63)
A typical approach to ensuring that the metadata attributes are appropriately backed up is to set up a mirror catalog, and use dynamic updates to the mirror catalog to maintain an active copy. This approach works as long as there are no errors in the original catalog.

To enable recovery from propagated errors, an independent snapshot of the catalog can be periodically created. This provides a second recovery mechanism in case both catalogs are compromised.
In addition to replication, the catalog indices need to be periodically optimized. This improves performance.

6.36 Set access approval flag (Policy 64)
This rule sets the ACCESS_APPROVAL flag to 1, and enables access by public and anonymous users. The rule uses the policy functions:

    addAVUMetadata
    checkCollInput
    deleteAVUMetadata

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-access-set.r.

```
allowAccess {
    # hipaa-access-set.r
    # The rule sets the ACCESS_APPROVAL flag to 1 on files in a collection
    # Public and anonymous access is enabled for all files in the collection
    # Get USER_ID corresponding to public and anonymous accounts
    checkCollInput (*Coll);
    *Qa = select USER_ID where USER_NAME = 'public';
    foreach(*Ra in *Qa) {*UserIdPublic = *Ra.USER_ID;}
    *Qb = select USER_ID where USER_NAME = 'anonymous';
    foreach(*Rb in *Qb) {*UserIdAnon = *Rb.USER_ID;}
    writeLine("stdout", "UserID for public is *UserIdPublic");
    writeLine("stdout", "UserID for anonymous is *UserIdAnon");
    *Q1 = select DATA_NAME, DATA_ID, COLL_NAME where COLL_NAME like '*Coll%';
    foreach (*R1 in *Q1) {
        *File = *R1.DATA_NAME;
        *FileId = *R1.DATA_ID;
        *Coll1 = *R1.COLL_NAME;
        *Path = "*Coll1/*File";
        *Q2 = select META_DATA_ATTR_NAME, META_DATA_ATTR_VALUE where COLL_NAME = 'Coll1' and DATA_NAME = '*File';
        *Count = 0;
        foreach(*R2 in *Q2) {
            *Name = *R2.META_DATA_ATTR_NAME;
            *Val = *R2.META_DATA_ATTR_VALUE;
            if(*Name == "ACCESS_APPROVAL") {
                *Count = *Count + 1;
                if(*Val != "1") {
                    # remove old ACCESS_APPROVAL flag
                    deleteAVUMetadata (*Path, "ACCESS_APPROVAL", *Val, ",", *Status);
                    # Set ACCESS_APPROVAL flag to 1
                    if (*Count == 1) {
                        addAVUMetadata (*Path, "ACCESS_APPROVAL", "1", ",", *Status);
                        writeLine("stdout", "Set ACCESS_APPROVAL for *Coll1/*File");
                    }
                }
            }
        }
    }
    if (*Count == 0) {
```
# Set ACCESS_APPROVAL flag to 1
    addAVUMetadata (*Path, "ACCESS_APPROVAL", "1", "", *Status);
    writeLine("stdout", "Set ACCESS_APPROVAL for *Coll1/*File");
}

# Check for public access on file
*Q3 = select count(DATA_ACCESS_USER_ID) where DATA_ACCESS_DATA_ID = '*FileId' and
    DATA_ACCESS_USER_ID = '*UserIdPublic';
foreach (*R3 in *Q3) {
    *Count = *R3.DATA_ACCESS_USER_ID;
    if (*Count == '0') {
        *File = *R1.DATA_NAME;
        *Path = "*Coll1/*File";
        msiSetACL("default", "read", "public", *Path);
        writeLine("stdout", "Enabled access for public in *File in *Coll1");
    }
}

*Q4 = select count(DATA_ACCESS_USER_ID) where DATA_ACCESS_DATA_ID = '*FileId' and
    DATA_ACCESS_USER_ID = '*UserIdAnon';
foreach (*R4 in *Q4) {
    *Count = *R4.DATA_ACCESS_USER_ID;
    if (*Count == '0') {
        *File = *R1.DATA_NAME;
        *Path = "*Coll1/*File";
        msiSetACL("default", "read", "anonymous", *Path);
        writeLine("stdout", "Enabled access for anonymous in *File in *Coll1");
    }
}

INPUT *Coll = "/UNC-CH/home/HIPAA/Archives"
OUTPUT ruleExecOut

6.36.1 Restrict access for “Protected” data

Each collection that contains “Protected” information will have an Approval flag, called

    ACCESS_APPROVAL

When the value of this attribute is set to “0”, no public or anonymous access is
allowed to files within the collection.

This rule sets the ACCESS_APPROVAL flag to 0 for every file in a collection, and
restricts access by public and anonymous accounts. The rule uses the policy
functions:

    addAVUMetadata
    checkCollInput
    deleteAVUMetadata

Updates to this policy are available from
http://github.com/DICE-UNC/policy-workbook/hipaa-restrict-access.r.

restrictAccess {
    # hipaa-restrict-access.r
    # The rule sets the ACCESS_APPROVAL flag to 0 on files in a collection
# Public and anonymous access is removed for all files in the collection
# Get USER_ID corresponding to public and anonymous accounts
checkCollInput (*Coll);
*Qa = select USER_ID where USER_NAME = 'public';
foreach(*Ra in *Qa) {*UserIdPublic = *Ra.USER_ID;}
*Qb = select USER_ID where USER_NAME = 'anonymous';
foreach(*Rb in *Qb) {*UserIdAnon = *Rb.USER_ID;}
writeLine("stdout", "UserID for public is *UserIdPublic");
writeLine("stdout", "UserID for anonymous is *UserIdAnon");
*Q1 = select DATA_NAME, DATA_ID, COLL_NAME where COLL_NAME like '*Coll%';
foreach (*R1 in *Q1) {
    *File = *R1.DATA_NAME;
    *FileId = *R1.DATA_ID;
    *Coll1 = *R1.COLL_NAME;
    *Path = "*Coll1/*File";
    *Q2 = select META_DATA_ATTR_NAME, META_DATA_ATTR_VALUE where COLL_NAME = '*Coll1' and DATA_NAME = '*File';
    *Count = 0;
    foreach(*R2 in *Q2) {
        *Name = *R2.META_DATA_ATTR_NAME;
        *Val = *R2.META_DATA_ATTR_VALUE;
        if(*Name == "ACCESS_APPROVAL") {
            *Count = *Count + 1;
            if(*Val != "0") {
                # remove old ACCESS_APPROVAL flag
                deleteAVUMetadata (*Path, "ACCESS_APPROVAL", *Val, ",", *Status);
            }
        }
    }
    if (*Count == 0) {
        # Set ACCESS_APPROVAL flag to 0
        addAVUMetadata (*Path, "ACCESS_APPROVAL", "0", ",", *Status);
        writeLine("stdout", "Set ACCESS_APPROVAL for *Coll1/*File");
    }
}
if (*Count == 0) {
    # Set ACCESS_APPROVAL flag to 0
    addAVUMetadata (*Path, "ACCESS_APPROVAL", "0", "", *Status);
    writeLine("stdout", "Set ACCESS_APPROVAL for *Coll1/*File");
}
# Check for public access on file
*Q3 = select DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = *FileId;
foreach (*R3 in *Q3) {
    *UserId = *R3.DATA_ACCESS_USER_ID;
    if (*UserId == *UserIdPublic) {
        *File = *R1.DATA_NAME;
        *Path = "*Coll1/*File";
        msiSetACL("default", "null", "public", *Path);
        writeLine("stdout", "Restrict access for public in *File in *Coll1");
    }
    if (*UserId == *UserIdAnon) {
        *File = *R1.DATA_NAME;
        *Path = "*Coll1/*File";
        msiSetACL("default", "null", "anonymous", *Path);
        writeLine("stdout", "Restrict access for anonymous in *File in *Coll1");
    }
}
6.37 Set access controls (Policy 14)
This rule keeps users from seeing the names of other user’s files. The rule sets the Access Control List policy. If the rule is not called or called with an argument other than STRICT, the STANDARD setting is in effect, which is fine for many sites. By default, users are allowed to see certain metadata, for example the data-object and sub-collection names in each other’s collections. When access controls are made STRICT by calling msiAclPolicy(STRICT), the General Query Access Control is applied on collections and data object metadata which means that the list command, ils, will need ‘read’ access or better to the collection to return the collection contents (name of data-objects, sub-collections, etc.).

The default is the normal, non-strict level, allowing users to see names of other collections. In all cases, access control to the data-objects is enforced. Even if a person can see file names in a collection, “read” access is required on a file to be able to read the file. Even with STRICT access control, however, the admin user is not restricted so various microservices and queries will still be able to evaluate system-wide information. The session variable, “$userNameClient” can be used to limit actions to individual users. However, this is only secure in an irods-password environment (not GSI), but you can then have rules for specific users:

```plaintext
acAclPolicy {ON($userNameClient == "quickshare") { }}
acAclPolicy {msiAclPolicy("STRICT"); }
```

which was requested by ARCS (Sean Fleming). See rsGenQuery.c for more information on $userNameClient. The typical use is to just set it strict or not for all users.

Updates to the rule are available at https://github.com/DICE-UNC/policy-workbook/blob/master/acAclPolicy-strict.re

```plaintext
acAclPolicy {msiAclPolicy("STRICT"); }
```

6.37.1 Set access controls after proprietary period
This rule checks a flag for whether a proprietary period has elapsed, and then provides public access to the file. The flag ACL_EXPIRY defines the date and time after which the file becomes public. The rule uses the policy function:

```plaintext
checkCollInput
```

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-set-ACL.r.

```plaintext
setACL {
    # hipaa-set-ACL.r
    # For each file with META_DATA_ATTR_NAME = ACL_EXPIRY
```
# Check value for whether period is over
# And set access to public and anonymous
msiGetSystemTime("Time", "unix");
*T = double(*Time);
*C = "/$rodsZoneClient/home/$userNameClient/" ++ *Coll;
checkCollInput(*Coll);
*Query = select DATA_NAME,META_DATA_ATTR_VALUE where COLL_NAME = *C and
META_DATA_ATTR_NAME = 'ACL_EXPIRY';
foreach(*Row in *Query) {
  *V = *Row.META_DATA_ATTR_VALUE;
  *Val = double(*V);
  *File = *Row.DATA_NAME
  *Path = *C ++ "/" ++ *File;
  if(*T > *Val) {
    # Time period has elapsed, provide public access
    msiSetACL("default", "read", "public", *Path);
    msiSetACL("default", "read", "anonymous", *Path);
    # Remove the metadata attribute
    *Str1 = "ACL_EXPIRY=*V";
    msiString2KeyValPair(*Str1, *kvp1);
    msiRemoveKeyValuePairsFromObj(*kvp1, *Path, "-d");
    writeLine("stdout", "Changed access to public for *Path");
  }
}
INPUT *Coll=$"test"
OUTPUT ruleExecOut

6.38 Set access restriction until approval flag is set (Policy 65)
When a file is added to a collection, it normally can only be accessed by the owner, the person uploading the file. The file can inherit access controls from its collection if the sticky bit is enabled. This applies the access controls from the collection as the access controls on the file.

A standard sequence is to:

- Turn off the inherit flag on the collection
- Load a file into the collection. The file can only be accessed by the owner of the file.
- Explicitly add access controls for a group
  - Members of the group can then access the file

When the approval flag is set to one, then public access can be enabled. Public access allows access by all accounts within the data grid. For access by persons without an account in the data grid, Anonymous access must also be enabled.

6.39 Set approval flag per collection for enabling bulk download (Policy 66)
Bulk downloads are initiated by a client, which manages either a loop over a specified file set or over files in a collection. Restriction of bulk download requires a policy enforcement point, acBulkGetPreProcPolicy. This could be turned off for in general.
The rule is available at

```
acBulkGetPreProcPolicy {msiSetBulkGetPostProcPolicy("off");}
```

Bulk processing can be turned off for a collection. Update to the rule are available at

```
acBulkGetPreProcPolicy{
    msiSplitPath($objPath,*Coll,*File);
    if (*Coll == "/UNC-CH/home/HIPAA") {
        msiSetBulkGetPostProcPolicy("off");
    }
}
```

Bulk processing can be controlled for a collection that has a flag “BulkDownload” with a value “off”. The rule is available at

```
acBulkGetPreProcPolicy {
    msiSplitPath($objPath,*Coll,*File);
    *Q1 = select META_COLL_ATTR_VALUE where COLL_NAME = *Coll and META_COLL_ATTR_NAME = 'BulkDownload';
    foreach (*R1 in *Q1) {*Val = *R1.META_COLL_ATTR_VALUE;}
    if (*Val == 'off') {msiSetBulkGetPostProcPolicy('off');}
}
```

These policies can be updated in the iRODS core.re file.

**6.40 Set asset protection classifier for data sets based on type of PII (Policy 67)**

Each data set should be assigned a protection classifier that defines whether the file contains:

- 1 - Protected Health Information – PHI
- 2 - Personally Identifiable Information – PII such as social security numbers
- 3 - Payment Card Information – PCI such as account numbers, card holder name, expiration date, service code, CID, PINs
- 4 - Legally restricted data – classified
- 5 - Proprietary information

The classifier is stored in a metadata attribute for each file:

- META_DATA_ATTR_NAME = AssetProtectionClassifier
- META_DATA_ATTR_VALUE = “protection classifier value 1-5”
- META_DATA_ATTR_UNIT = “”

An approach is to use a bitcurator rule to assign asset classifier for PII, PHI, PCI.

**6.41 Set flag for whether tickets can be used on files in a collection (Policy 68)**

The iRODS data grid supports the creation of tickets that enable access to specific data sets by persons who do not have an account. The tickets control the number of allowed accesses and the time period during which the access can be made. For
collections that have the ACCESS_APPROVAL flag set to 0, ticket-based access is prohibited.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acTicketPolicy.re.

```plaintext
acTicketPolicy {
    # For collections that have the ACCESS_APPROVAL flag set to 0, tickets are disabled
    msiSplitPath($objPath, *Coll, *File);
    *Q = select META_COLL_ATTR_VALUE where COLL_NAME = '*Coll' and
       META_COLL_ATTR_NAME = 'ACCESS_APPROVAL';
    *Access == "0";
    foreach (*R in *Q) {
        *Access = *R.META_COLL_ATTR_VALUE;
    }
    if (*Access == "0") {
        writeLine("serverlog", "Restrict ticket access for collection *Coll and file *File");
        fail;
    }
}
```

6.41.1 Remove public and anonymous access
Ticket access requires that anonymous access permission be set. When the ACCESS_APPROVAL flag is set to 0, anonymous access is turned off. Thus ticket access can be controlled by setting the ACCESS_APPROVAL flag to 0. The rule listed in section 6.36.1 can be used to set the ACCESS_APPROVAL flag to 0.

6.42 Set lockout flag and period on user name - counting number of tries (Policy 69)
When a user exceeds the number of allowed attempts when trying to log on without success, a lockout flag will be set for a specified period of time. Ideally this is done by the authentication system.

6.42.1 Set lockout period on user name
The code that checks the user name will need to be augmented with a policy enforcement point (acChkUserLogon) that implements three metadata attributes for a user:

- META_USER_ATTR_NAME = NumberAttempts
- META_USER_ATTR_NAME = LockoutPeriod
- META_USER_ATTR_NAME = ResetPassword

The control point acChkUserLogon will need to be called for every controlled iCommand. Note that the NumberAttempts counter will need to be set back to “0” on a successful login.

This rule sets increments the attempt counter, and sets an expiration time when the allowed number of attempts is exceeded.
Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acChkUserLogon.re.

    acChkUserLogon {
    # Increment a counter for each logon attempt
    *User = $userNameClient;
    *Q1 = select META_USER_ATTR_VALUE where USER_NAME = '*User' and
      META_USER_ATTR_NAME = 'NumberAttempts';
    foreach (*R1 in *Q1) {
      *Val = *R1.META_USER_ATTR_VALUE;
      *Str = "NumberAttempts=*Val";
      msiString2KeyValPair(*Str,*Kvp);
      msiRemoveKeyValuePairsFromObj(*Kvp,*User,"-u");
      *Val1 = str(int(*Val) + 1);
      *Str1 = "NumberAttempts=*Val1";
      msiString2KeyValPair(*Str1,*Kvp1);
      msiAssociateKeyValuePairsToObj(*Kvp1,*User,"-u");
    }
    if (int(*Val1) > 5) {
      # set lockout period
      *Q2 = select META_USER_ATTR_VALUE where USER_NAME = '*User' and
        META_USER_ATTR_NAME = 'LockoutPeriod';
      foreach (*R2 in *Q2) {
        *Val = *R2.META_USER_ATTR_VALUE;
        *Str = "LockoutPeriod=*Val";
        msiString2KeyValPair(*Str,*Kvp);
        msiRemoveKeyValuePairsFromObj(*Kvp,*User,"-u");
        msiGetSystemTime(*Tim,"unix");
        *Str1 = "LockoutPeriod=*Tim";
        msiString2KeyValPair(*Str1,*Kvp1);
        msiAssociateKeyValuePairsToObj(*Kvp1,*User,"-u");
      }
    }
    }

A second rule tests the expiration time to release the lockout flag. This rule could be added to the acSetPublicUserPolicy.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acSetPublicUserPolicy-lockout.re.

    acSetPublicUserPolicy {
    # reset NumberAttempts and LockoutPeriod
    *User = $userNameClient;
    *Q1 = select META_USER_ATTR_VALUE where USER_NAME = '*User' and
      META_USER_ATTR_NAME = 'NumberAttempts';
    foreach (*R1 in *Q1) {
      *Val = *R1.META_USER_ATTR_VALUE;
      *Str = "NumberAttempts=*Val";
      msiString2KeyValPair(*Str,*Kvp);
      msiRemoveKeyValuePairsFromObj(*Kvp,*User,"-u");
    }
    # reset LockoutPeriod
    *Q2 = select META_USER_ATTR_VALUE where USER_NAME = '*User' and
      META_USER_ATTR_NAME = 'LockoutPeriod';
    foreach (*R2 in *Q2) {
      *Val = *R2.META_USER_ATTR_VALUE;
      *Str = "LockoutPeriod=*Val";
      msiString2KeyValPair(*Str,*Kvp);
      msiRemoveKeyValuePairsFromObj(*Kvp,*User,"-u");
      msiGetSystemTime(*Tim,"unix");
      *Str1 = "LockoutPeriod=*Tim";
      msiString2KeyValPair(*Str1,*Kvp1);
      msiAssociateKeyValuePairsToObj(*Kvp1,*User,"-u");
    }
    }
Str = "NumberAttempts=*Val";
msiString2KeyValPair(*Str,*Kvp);
msiRemoveKeyValuePairsFromObj(*Kvp,*User, "-u");
*Val1 = "0";
*Str1 = "NumberAttempts=*Val1";
msiString2KeyValPair(*Str1,*Kvp1);
msiAssociateKeyValuePairsToObj(*Kvp1,*User, "-u");
}
if (int(*Val1) > 5) {
# set lockout period
*Q2 = select META_USER_ATTR_VALUE where USER_NAME = '*User' and
META_USER_ATTR_NAME = 'LockoutPeriod';
foreach (*R2 in *Q2) {
  *Val = *R2.META_USER_ATTR_VALUE;
  *Str = "LockoutPeriod=*Val";
  msiString2KeyValPair(*Str,*Kvp);
  msiRemoveKeyValuePairsFromObj(*Kvp,*User, "-u");
  *Str1 = "LockoutPeriod=0";
  msiString2KeyValPair(*Str1,*Kvp1);
  msiAssociateKeyValuePairsToObj(*Kvp1,*User, "-u");
}
}

6.43 Set password update flag on user name (Policy 70)
A flag is associated with each user name to specify whether they need to update their password. This uses the attribute:

- META_USER_ATTR_NAME = ResetPassword

The value can be set to '1' for all users by the administrator.

Updates to this policy are available from

passwordUpdate {
  # hipaa-password-update.r
  # All users have a password update flag set
  *Q1 = select USER_NAME;
  foreach (*R1 in *Q1) {
    *User = *R1.USER_NAME;
    if (*User != 'public' && *User != 'anonymous') {
      *Q2 = select META_USER_ATTR_VALUE where USER_NAME = '*User' and
      META_USER_ATTR_NAME = 'ResetPassword';
      foreach (*R2 *Q2) {
        *Val = *R2.META_USER_ATTR_VALUE;
        *Str = "ResetPassword=*Val";
        msiString2KeyValPair(*Str,*Kvp);
        msiRemoveKeyValuePairsFromObj(*Kvp,*User, "-u");
        *Val1 = "1";
      }
    }
  }
}
*Str1 = "ResetPassword=*Val1";
msiString2KeyValPair(*Str1,*Kvp1);
msiAssociateKeyValuePairsToObj(*Kvp1,*User,"-u");
}
}
}
}
INPUT null
OUTPUT ruleExecOut

Each time the acSetPublicPolicy enforcement point is executed, the ResetPassword flag can be checked and a message can be written to stdout.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acSetPublicUserPolicy-reset.re.

```plaintext
acSetPublicUserPolicy {
  # reset NumberAttempts and LockoutPeriod
  *User = $userNameClient;
  *Q1 = select META_USER_ATTR_VALUE where USER_NAME = "*User" and
      META_USER_ATTR_NAME = 'NumberAttempts';
  foreach (*R1 in *Q1) {
    *Val = *R1.META_USER_ATTR_VALUE;
    *Str = "NumberAttempts=*Val";
    msiString2KeyValPair(*Str,*Kvp);
    msiRemoveKeyValuePairsFromObj(*Kvp,*User,"-u");
    *Val1 = "0";
    *Str1 = "NumberAttempts=*Val1";
    msiString2KeyValPair(*Str1,*Kvp1);
    msiAssociateKeyValuePairsToObj(*Kvp1,*User,"-u");
  }
  if (int(*Val1) > 5) {  
    # set lockout period
    *Q2 = select META_USER_ATTR_VALUE where USER_NAME = "*User" and
        META_USER_ATTR_NAME = 'LockoutPeriod';
    foreach (*R2 in *Q2) {
      *Val = *R2.META_USER_ATTR_VALUE;
      *Str = "LockoutPeriod=*Val";
      msiString2KeyValPair(*Str,*Kvp);
      msiRemoveKeyValuePairsFromObj(*Kvp,*User,"-u");
      *Str1 = "LockoutPeriod=0";
      msiString2KeyValPair(*Str1,*Kvp1);
      msiAssociateKeyValuePairsToObj(*Kvp1,*User,"-u");
    }
  }
  if (*User != 'public' && *User != 'anonymous') {  
    *Q2 = select META_USER_ATTR_VALUE where USER_NAME = "*User" and
        META_USER_ATTR_NAME = 'ResetPassword';
    foreach (*R2 in *Q2) {
```
6.44 Set retention period for data reviews (Policy 71)
The iRODS data grid provides a metadata attribute, DATA_EXPIRY, for a retention period. The choice of what to do when the retention period is over is governed by a disposition policy. One approach is to set DATA_EXPIRY for a data review. A query can then be issued to identify files that need to be reviewed. The rule uses the policy function:

\[
\text{checkCollInput}
\]

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-retention-review.r.

```r
retentionReview {
    # hipaa-retention-review.r
    # list all files that have exceeded their retention period
    # this assumes DATA_EXPIRY is initialized with the unix time
    checkCollInput(*Coll);  
    msiGetSystemTime("Time", 'unix');  
    msiGetSystemTime("Timh", 'human');  
    *Q1 = select DATA_NAME, COLL_NAME where COLL_NAME like 'Coll%' and DATA_EXPIRY <= "Time";  
    writeLine("stdout", "Review status of files with expiration date <= "Timh");  
    foreach (*R1 in *Q1) {  
        *File = *R1.DATA_NAME;  
        *Coll = *R1.COLL_NAME;  
        writeLine("stdout", "Check *Coll/*File");  
    }  
}
```

**INPUT** *Coll = "/UNC-HIPAA/home/HIPAA/Archive"
**OUTPUT** ruleExecOut

6.45 Set retention period on ingestion (Policy 21)
A system attribute, DATA_EXPIRY, is used to define an expiration date for a digital object. This rule sets an expiration date a specified number of seconds greater than the ingestion time for a specified collection

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acPostProcForPut-expiry.re.

```r
acPostProcForPut {  
    msiSplitPath($objPath, *Coll, *File);  
    if(*Coll == "/UNC-ARCHIVE/home/Archive") {  
        msiGetSystemTime("T", 'unix');  
        *Time = int(*T) + 3600*24*365;  
        msiSysMetaModify("expirytime", "*Time");  
    }
```
6.46 Track systems by type (server, laptop, router,..) (Policy 72)
Each system used within the repository can be labeled by its type. The information can be kept in a file that is stored in the Reports folder. This policy defines the collection location and file name used for the report.

- Technology report name: LogSystemType
- Collection name: Reports
- Location: /UNC-CH/home/HIPAA/Reports

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-store-system-log.r.

```r
storeSystemLog {
  # hipaa-store-system-log.r
  # Stores report in /UNC-CH/home/HIPAA/Reports
  # Saves copy on local disk with the name LogSoftwareChanges
  *Coll = "/UNC-CH/home/HIPAA/Reports";
  *File = "LogSystemType";
  *Path = "*Coll/*File";
  msiDataObjPut(*Path, "destRescName","localPath=+/File++++forceFlag=", *Status);
}
```

6.47 Verify approval flags within a collection (Policy 73)
This rule examines a collection to determine whether any of the files have not been approved for access, and lists all such files. The rule uses the policy function: checkCollInput

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-check-access-approval.r.

```r
setAccessApproval {
  # hipaa-check-access-approval.r
  # The rule checks the ACCESS_APPROVAL flag on files in a collection
  checkCollInput (*Coll);
  *Q1 = select DATA_NAME, DATA_ID, COLL_NAME where COLL_NAME like "*Coll%";
  foreach (*R1 in *Q1) {
    *File = *R1.DATA_NAME;
    *FileId = *R1.DATA_ID;
    *Coll1 = *R1.COLL_NAME;
    *Path = "*/Coll/*File";
    *Q2 = select META_DATA_ATTR_NAME, META_DATA_ATTR_VALUE where COLL_NAME = *Coll1 and DATA_NAME = *File;
    *Count = 0;
    foreach (*R2 in *Q2) {
      *Name = *R2.META_DATA_ATTR_NAME;
      *Val = *R2.META_DATA_ATTR_VALUE;
      if(*Name == "ACCESS_APPROVAL") {
```
*Count = 1;
if(*Val != "0") {
    writeLine("stdout","Path has ACCESS_APPROVAL != 0");
    break;
}
}
if (*Count == 0) {
    writeLine("stdout", "Path has no ACCESS_APPROVAL");
}
}

INPUT *Coll = "/UNC-CH/home/HIPAA/Archives"
OUTPUT ruleExecOut

6.48 Verify files have not been corrupted (Policy 18)
The rule for verifying that files have not been corrupted can be combined with the rule to check existence of replicas. A version of the rule is listed in section 4.5.2.

6.49 Verify presence of required replicas (Policy 74)
A rule can be run periodically to verify that every file has a replica. This rule checks both the existence of the required replica, validates the checksums, and replaces missing or corrupted files. A version of the rule is listed in section 4.5.2.

6.50 Verify that no controlled data have public or anonymous access (Policy 75)
Each collection that contains “Protected” information will have an Approval flag, called

ACCESS_APPROVAL
When the value of this attribute is set to “0”, no public or anonymous access is allowed to files within the collection. When the flag is set to “1”, anonymous access is allowed.

6.50.1 Restrict access to “Protected” data
This rule checks the ACCESS_APPROVAL flag, and restricts access by public and anonymous accounts.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/hipaa-verify-access-approval.r.

AccessCheck {
    # hipaa-verify-access-approval.r
    # The rule checks the ACCESS_APPROVAL flag on each collection
    # If the value is "0", public and anonymous access is removed for all files in the collection
    # Get USER_ID corresponding to public and anonymous accounts
    *Qa = select USER_ID where USER_NAME = 'public';
    foreach(*Ra in *Qa) {*UserIdPublic = *Ra.USER_ID;}
    *Qb = select USER_ID where USER_NAME = 'anonymous';
    foreach(*Rb in *Qb) {*UserIdAnon = *Rb.USER_ID;}
    writeLine("stdout", "UserID for public is *UserIdPublic");
    writeLine("stdout", "UserID for anonymous is *UserIdAnon");
    # get all collection names where ACCESS_APPROVAL is set
    *Q = select COLL_NAME where META_COLL_ATTR_NAME = "ACCESS_APPROVAL" and
META_COLL_ATTR_VALUE = '0';
foreach (*R in *Q) {
    *Coll = *R.COLL_NAME;
    # Get list of files in the collection
    *Q1 = select DATA_ID, DATA_NAME where COLL_NAME = '*Coll';
    foreach(*R1 in *Q1) {
        *FileId = *R1.DATA_ID;
        # Check for public access on file
        *Q2 = select DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = '*FileId';
        foreach (*R2 in *Q2) {
            *UserId = *R2.DATA_ACCESS_USER_ID;
            if (*UserId == *UserIdPublic) {
                *File = *R1.DATA_NAME;
                *Path = "*Coll/*File"	;
                msiSetACL("default", "null", "public", *Path);
                writeLine("stdout", "Reset access for public in *File in *Coll");
            }
            if (*UserId == *UserIdAnon) {
                *File = *R1.DATA_NAME;
                *Path = "*Coll/*File"	;
                msiSetACL("default", "null", "anonymous", *Path);
                writeLine("stdout", "Reset access for anonymous in *File in *Coll");
            }
        }
    }
}
INPUT null
OUTPUT ruleExecOut

6.51 Verify that protected assets have been encrypted (Policy 76)
Check that all files in the collection
/UNC-CH/home/HIPAA/Archive
have the DATA_ENCRYPT flag set to 1. If the flag is missing or the value is not 1,
write an output line and encrypt the file.

6.51.1 Check that files with ACCESS_APPROVAL = 0 are encrypted
This version of the rule looks for the ACCESS_APPROVAL flag. If the value is set to 0,
then the file encryption is checked. If the file is not encrypted, an output line is
written and the file is encrypted.

Updates to this policy are available from

encryptCheck {
    # hipaa-encrypt-check.r
    # Check that all files with ACCESS_APPROVAL = 0 have been encrypted
    # and encrypt files if needed
    *Q1 = select DATA_NAME, COLL_NAME where META_DATA_ATTR_NAME = 'ACCESS_APPROVAL' and META_DATA_ATTR_VALUE = '0';
    foreach (*R1 in *Q1) {
        *File = *R1.DATA_NAME;
*Coll = *R1.COLL_NAME;
*Q2 = select META_DATA_ATTR_VALUE, META_DATA_ATTR_NAME where COLL_NAME = '*Coll' and DATA_NAME = '*File';
*Count = 0;
foreach (*R2 in *Q2) {
  *Name = *R2.META_DATA_ATTR_NAME;
  *Val = *Rw.META_DATA_ATTR_VALUE;
  if (*Name == "DATA_ENCRYPT") {
    *Count = 1;
    if (*Val != "1") {
      writeLine("stdout", "File *File has not been encrypted");
      *Path = "*Coll/*File";
      msiEncrypt(*Path);
      writeLine("stdout", "File *File has been encrypted");
      # remove old encrypt flag
      *Str0 = "DATA_ENCRYPT=*Val";
      msiString2KeyValPair(*Str0, *Kvp0);
      msiRemoveKeyValuePairsFromObj(*Kvp0, *Path, "-d");
      # Set encrypt flag to 1
      *Str1 = "DATA_ENCRYPT=1";
      msiString2KeyValPair(*Str1, *Kvp1);
      msiAssociateKeyValuePairsToObj(*Kvp1, *Path, "-d");
    }
  }
} if (*Count == 0) {
  # encrypt
  writeLine("stdout", "File *File has not been encrypted");
  *Path = "*Coll/*File";
  msiEncrypt(*Path);
  writeLine("stdout", "File *File has been encrypted");
  # Set encrypt flag to 1
  *Str1 = "DATA_ENCRYPT=1";
  msiString2KeyValPair(*Str1, *Kvp1);
  msiAssociateKeyValuePairsToObj(*Kvp1, *Path, "-d");
} INPUT null
OUTPUT ruleExecOut
7 Data Management Plan Example Rules

Data management plans (DMPs) are required by the National Science Foundation and other federal agencies for every submitted proposal. The DMPs specify tasks related to formation of the digital collection, analysis, storage, publication, and archives. The expectation is that the tasks can be automated through policies that are either applied at policy enforcement points, or that are periodically executed.

An analysis of NSF requirements for DMPs is shown in Table 7.1. A total of 38 tasks were identified, along with the type of environment variable needed as input for each task.

<table>
<thead>
<tr>
<th>Table 7.1. Data Management Plan Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMP tasks</td>
</tr>
<tr>
<td>----------------------------------</td>
</tr>
<tr>
<td>1 Collection</td>
</tr>
<tr>
<td>2 Costs</td>
</tr>
<tr>
<td>3 Collection plans</td>
</tr>
<tr>
<td>4 Instrument types</td>
</tr>
<tr>
<td>5 Event log</td>
</tr>
<tr>
<td>6 Collection report</td>
</tr>
<tr>
<td>7 Required data policies</td>
</tr>
<tr>
<td>8 Data category</td>
</tr>
<tr>
<td>9 Use of existing data</td>
</tr>
<tr>
<td>10 Analysis</td>
</tr>
<tr>
<td>11 Analysis plans</td>
</tr>
<tr>
<td>12 Data sharing during analysis</td>
</tr>
<tr>
<td>13 Data dictionary / glossary</td>
</tr>
<tr>
<td>14 Naming includes</td>
</tr>
<tr>
<td>15 Data format type</td>
</tr>
<tr>
<td>16 DOI for data sets</td>
</tr>
<tr>
<td>17 Metadata standard</td>
</tr>
<tr>
<td>18 Metadata export as</td>
</tr>
<tr>
<td>19 Storage</td>
</tr>
<tr>
<td>20 Size</td>
</tr>
<tr>
<td>21 Publication</td>
</tr>
<tr>
<td>22 Make Data products public</td>
</tr>
<tr>
<td>23 Re-use</td>
</tr>
<tr>
<td>24 Re-distribution</td>
</tr>
<tr>
<td>25 Access restrictions</td>
</tr>
<tr>
<td>26 IPR</td>
</tr>
<tr>
<td>27 Web access through</td>
</tr>
<tr>
<td>28 Data sharing system</td>
</tr>
<tr>
<td>29 Code distribution system</td>
</tr>
<tr>
<td>30 Archive</td>
</tr>
<tr>
<td>31 Curation</td>
</tr>
<tr>
<td>32 Archive</td>
</tr>
<tr>
<td>33 Number of replicas</td>
</tr>
<tr>
<td>34 Backup frequency</td>
</tr>
<tr>
<td>35 Integrity check frequency</td>
</tr>
<tr>
<td>36 Technology evolution</td>
</tr>
<tr>
<td>37 Catalog</td>
</tr>
<tr>
<td>38 Transformative migration</td>
</tr>
</tbody>
</table>

Each directorate and division at NSF has selected different aspects to emphasize. These preferred tasks are indicated in Table 7.2.
Table 7.2. DMP tasks by NSF Directorate/Division

<table>
<thead>
<tr>
<th></th>
<th>AGS</th>
<th>AST</th>
<th>CHE</th>
<th>CISE</th>
<th>DMR</th>
<th>EAR</th>
<th>EHR</th>
<th>ENG</th>
<th>GEN</th>
<th>OCE</th>
<th>PHY</th>
<th>SBE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>16</td>
<td></td>
<td>Cite</td>
<td>URL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>21</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>&lt;2 yrs</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>22</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>23</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>24</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>25</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>26</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>27</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>28</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>31</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>32</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>33</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>34</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>35</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>36</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>38</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

To understand how actual DMPs were created, 18 Data Management Plans (DMP) were compared to determine whether a common set of policies could be implemented for automating management tasks. The DMPs were acquired from the DataONE web site (example DMPs) and from the Data Management Planning tool (public DMPS from the California Digital Library). Each DMP was compared with the tasks determined from the NSF requirements.

The expectation is that each task can be automated by creating a set of data management policies for setting environment variables (such as retention period), enforcing the policy, and verifying the policy. The tasks from the DMPs are listed in Tables 7.3A and 7.3B. The tasks specified in the DMPs varied dramatically. For the tasks that depended upon an environmental variable, the value of the variable was specified for each task for each plan.
Table 7.3A – Published data management plans

<table>
<thead>
<tr>
<th>Task</th>
<th>Environment Variables</th>
<th>Cultural Objects</th>
<th>Mauna Loa CO2 Sensor</th>
<th>Surface weather data</th>
<th>Multimedia Text Annotation</th>
<th>Parietal Cortex</th>
<th>Biosignature Suites</th>
<th>Peer Power</th>
<th>Anthropod Responses</th>
<th>Andvarri</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NEH</td>
<td>AGS</td>
<td>AGS</td>
<td>NEH</td>
<td>BIO</td>
<td>BIO</td>
<td>CISE</td>
<td>GEN</td>
<td>NEH</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Roles</td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Budget</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>How, what</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Type</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Event</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Event</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Products</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Source</td>
<td></td>
<td></td>
<td></td>
<td>GHCN-D</td>
<td></td>
<td></td>
<td></td>
<td>Institutions</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Plans</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Who</td>
<td></td>
<td></td>
<td></td>
<td>Generate netCDF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Attributes</td>
<td>timestamp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Type</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>timestamp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Type</td>
<td>.txt</td>
<td>CSV, text</td>
<td>CSV, netCDF</td>
<td>.plx, .dvt, .avt, .pdf, .tif, .csv, .txt, .xls, .csv</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Type</td>
<td>URL</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Type</td>
<td>METS</td>
<td>X</td>
<td>X</td>
<td>generate netCDF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Location</td>
<td>XML</td>
<td>XML</td>
<td>images</td>
<td>Dublin Core, EML</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>When</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Policies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Community</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Privacy</td>
<td>none</td>
<td>creator, copyright</td>
<td></td>
<td>CCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Type</td>
<td>none</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>How</td>
<td>URLs</td>
<td>URLs</td>
<td>URLs</td>
<td>FTP, website</td>
<td>URL</td>
<td>website</td>
<td>URL, website</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Type</td>
<td>Dspace</td>
<td>iRODS</td>
<td>google docs</td>
<td>Dspace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Type</td>
<td>UCSC</td>
<td>GitHub</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Period</td>
<td>forever</td>
<td>forever</td>
<td>5 yr</td>
<td>10 yr</td>
<td>forever</td>
<td>long-term project</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Plans</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Location</td>
<td>UC3</td>
<td>ORNL</td>
<td>ORNL</td>
<td>IDEC, CCNP, UNM-Dspace</td>
<td>mySQL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>#</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Policies</td>
<td>Daily</td>
<td>Daily, monthly</td>
<td>periodic</td>
<td>periodic</td>
<td>daily</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Policies</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Plans</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Metadata</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Metadata</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

167
## Table 7.3B – Data Management Plans

<table>
<thead>
<tr>
<th>Task</th>
<th>Making Data Count</th>
<th>Collaboration as a means of retention</th>
<th>Meteorological measurements in Eastern Antarctica</th>
<th>Project 1 data management</th>
<th>Agent-based model of population</th>
<th>Engineered Bioactive Interfaces</th>
<th>Inquiring into Engineering</th>
<th>Certain Stem</th>
<th>HydroShare</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>AGS</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>DataONE</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td>.txt, .csv</td>
<td>.html, .css, .xml</td>
<td>ArcGIS .tif, .txt</td>
<td></td>
<td>audio, .txt, .xsl</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>EZID</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>COUNTER</td>
<td>WMO</td>
<td>FGDC</td>
<td></td>
<td></td>
<td>X</td>
<td>education</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td>.txt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Apache 2</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>NSF</td>
<td>Collaboration driven</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td>project</td>
<td></td>
<td></td>
<td></td>
<td>project</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Use agreement</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>IRB</td>
<td>IRB</td>
<td>proprietary</td>
<td>IRB IRB</td>
<td></td>
<td>Research group driven</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td>website</td>
<td>website</td>
<td>website</td>
<td></td>
<td>HTTP, FTP, DataONE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>Merritt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CUAHSI HIS, HUBzero, iRODS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>gForge</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td>10 yrs</td>
<td>3 yrs</td>
<td>3 yrs</td>
<td>10 yrs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>GitHub</td>
<td>OSF.io</td>
<td>US ADC</td>
<td>EPA</td>
<td>Unknown</td>
<td>Data Commons</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td>3</td>
<td>months</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Task 27 specified the type of access client that could be used to interact with the data collection. Most DMPs planned to publish data through a local web site or to provide persistent URLs to enable remote access to the data sets.

Task 30 specified the retention period. Some sites planned to keep the data forever, or as long as the designated repository was functional.

Task 32 specified the repository where the data sets would be managed. The DMPs identified a wide variety of data management systems, from local disk caches, to local databases, to institutional repositories, to federal repositories. Most of the DMPS did not specify the resources where the collection would be assembled, and instead specified the final archive.

The most comprehensive DMP plan that was examined was the DataONE example Data Management Plan for “Atmospheric Concentrations, Mauna Loa Observatory, Hawaii, 2011-2013”. This plan included 16 of the policies. The Mauna Loa DMP is listed in Appendix F. We analyzed the plan to identify the data management requirements and extracted the following tasks:

3. Plans for assembling the collection
5. Maintenance of an event log recording changes to sensors
6. Maintenance of a collection report
8. Categorization as observational data
10. Quality assessment
11. Analysis plans
14. Timestamp included in file name
15. Data types are .csv, .txt
16. DOI created for each file
17. Metadata standard based on discipline
18. Metadata exported as XML
21. All original data is made public
22. Data products are made public after 6 months and review
27. Web access provided through URLs
30. Data retained forever
32. Data archived at ORNL

A similar analysis was done for administration of protected data at UNC including PII, PHI, and PCI data types. A total of 48 tasks was identified, including password strength assessments, detection of the presence of protected data, characterization of the type of protected data, logging of access events, and analysis of audit trails. This indicated that the task list for data management plans is expected to expand as additional types of data are managed.

For each task, we create a computer actionable rule that can be used to automate execution. We use the integrated Rule Oriented Data System rule language to write the rules. The resulting rules are listed below for each task.
### 7.1 Staffing policies (Policy 48)

The roles needed to implement a data management plan include:

1. administrator – person making the financial commitment for maintaining the repository
2. collection manager – person maintaining the properties of the data collection (required metadata and data format standards, collection quality)
3. data grid administrator – person maintaining the properties of the repository (repository software upgrades, drivers for storage systems, clients)
4. information technology administrator – person maintaining the storage systems, network, authentication systems.

Typically, at least two persons are needed for each of the data grid and information technology administrator positions. This provides redundancy needed to ensure access across vacations.

The following policy counts the number of data grid administrators for a collection. The policy checks the number of users who can access a specified collection and lists their account names.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-list-admin.r.

```r
listAdmin {
  # dmp-list-admin.r
  # list all persons with rodsadmin status
  *Count = 0;
  *Q1 = select USER_NAME where USER_TYPE = 'rodsadmin';
  foreach (*R1 in *Q1) {
    *Name = *R1.USER_NAME;
    *Count = *Count + 1;
    writeLine("stdout","Person with admin privilege *Name");
  }
  writeLine("stdout","*Count persons have admin privilege");
}
```

### 7.2 Cost reporting (Policy 24)

The cost of managing a data collection includes:

1. Facility costs for floor space and power
2. Equipment costs for storage systems, networks, and computer servers
3. Media costs for tape
4. Labor costs for operations
5. Network costs for loading the collection and for collection access

The costs can be distributed across the files in the collection. However, the costs may be proportional to:

- The number of files
- The size of the files
The amount of metadata
A policy that aggregates costs across these three metrics is listed below. The rule uses the policy functions:
  checkCollInput
  checkRescInput
  createLogFile
  findZoneHostName
  isColl

Updates to this policy are available from
http://github.com/DICE-UNC/policy-workbook/dmp-cost.r.

costReport {
  # dmp-cost.r
  # calculate cost based on number of files, size, and metadata
  # tabulate costs for all files in the *Src directory
  # report is stored in specified directory, *Rep
  checkCollInput (*Src);
  checkCollInput (*Rep);
  checkRescInput (*Res, $rodsZoneClient);
  #============ create a collection for log files if it does not exist ===============

  *Count = 0;
  *Tot = 0.;
  *Countmeta = 0;
  *Q = select DATA_ID, DATA_SIZE where COLL_NAME like '*Src';
  foreach (*R in *Q) {
    *Count = *Count + 1;
    *Size = *R.DATA_SIZE;
    *DataID = *R.DATA_ID;
    *Tot = *Tot + double(*Size);
    *Q1 = select count(META_DATA_ATTR_ID) where DATA_ID = *DataID;
    foreach (*R1 in *Q1) {*Cmeta = *R1.META_DATA_ATTR_ID;}
    *Countmeta = *Countmeta + int(*Cmeta);
  }

  *CostS = *FacSize * *Tot / 1000000000.;
  *CostN = *FacCount * *Count / 1000000;
  *CostM = *FacMeta * *Countmeta / 1000000;
  writeLine("*Lfile", "Storage cost = \$\*CostS for *Tot bytes");
  writeLine("*Lfile", "File cost = \$\*CostN for *Count files");
  writeLine("*Lfile", "Metadata cost = \$\*CostM for *Countmeta attributes");
}

INPUT *FacSize=0.10, *FacCount=1., *FacMeta=1.,
       *Src="/sandbox/home/$userNameClient/archive", *Res="demoResc",
       *Rep="/sandbox/home/$userNameClient/reports"
OUTPUT ruleExecOut

7.3 Collection creation planning (Policy 45)
Collection creation planning identifies the properties that will be associated with a collection. The properties are driven by assertions that the collection creators will
claim about the digital entities, such as provenance, authenticity, quality, completeness. Collection planning also requires the identification of:
- Mechanisms for ingesting sensor data into a collection
- Naming conventions assigned to the files
- Arrangement of files into collection
- Identification of appropriate provenance metadata
- Identification of appropriate description metadata
- Assignment of access controls
- Identification of procedures for generating derived data products.
- Quality control

The specific policies that automate these tasks depend upon the specific details of the collection formation process and the type of data that are being organized (observational, experimental, simulation, survey). Example policies for collection arrangement might be:
- Organize by time period. Each month a new subcollection is started.
- Organize by data type. Separate collections are made for sensor data, simulation data, documents.
- Organize by contributor.
- Organize by experiment.

The example policy listed below organizes data files by a time extension. Files are copied from a staging area into subcollections for each year. The rule uses the policy functions:
  checkCollInput
  isColl

Updates to this policy are available from
http://github.com/DICE-UNC/policy-workbook/dmp-stage-time.r.

```r
stageByTime {
    # dmp-stage-time.r
    # copy files from a staging area into a new subdirectory each year
    # extract year as extension from each file
    # sort files into yearly subdirectories
    checkCollInput("Srccoll");
    checkCollInput("Destcoll");
    *Q1 = select DATA_NAME where COLL_NAME = "Srccoll";
    foreach (*R1 in *Q1) {
        *File = *R1.DATA_NAME;
        msiSplitPathByKey(*File, ".", *head, *B);
        *Src = "Srccoll ++ "/"File";
        *Destc = "Destcoll ++ "/"B";
        isColl(*Destc, "stdout", *Status);
        if (*Status >= 0) {
            *Dest = "Destc ++ "/"File"
            msiDataObjRename(*Src, *Dest, "0", *Status);
            writeLine("stdout", "moved file from *Src to *Dest");
        }
    }
}
```
7.4 Instrument control (Policy 77)

The control of the data streams from sensors requires identification of how frequently to harvest observational data, how to aggregate the sensor data into files, and how to archive the data streams. As an example, we illustrate the harvesting of sensor data from an external Antelope Real Time System. The planning requires identifying how frequently to harvest, the format to be used to store the data, and how to name the files. The rule harvests 100,000 packets from a specific sensor.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-sensor-harvest.r.

antelopRule{
    # dmp-sensor-harvest.r
    # Store Packet in NetCDF CDL format=100
    *SColl = *Coll ++ "/" ++ *Sensor
    *SFile = *SColl ++ "/" ++ "waveform.cdl"
    msiCollCreate(*SColl,"1",*STAT_1);
    openForAppendOrCreate(*SFile,*Resc,*D_FD,*New);
    # Get Packet, Reap, Decode and Store
    msiOrbOpen(*orbHost,*orbParam,*orbId);
    msiOrbSelect(*orbId,*Sensor,*sresOut);
    msiOrbReap(*orbId,*pktId,*srcName,*oTime,*pktOut,*nBytes,*resOut);
    msiOrbDecodePkt(*orbId,*modeIn + *New,*srcName,*oTime,*pktOut,*nBytes,*decodeBufInOut);
    msiDataObjWrite(*D_FD,*decodeBufInOut,*WR_LN);
    for(*I=0;*I<*PKTNum;*I=*I+1) {
        msiOrbReap(*orbId, *pktId2, *srcName2, *oTime2, *pktOut2, *nBytes2, *resOut2);
        msiOrbDecodePkt(*orbId, *modeIn, *srcName2, *oTime2, *pktOut2, *nBytes2, *decodeBufInOut2);
        msiDataObjLseek(*D_FD,*Offset,*Loc,*Status2);
        msiDataObjWrite(*D_FD,",\n",*WR_LN_F4);
        msiDataObjWrite(*D_FD,*decodeBufInOut2,*WR_LN2);
        msiDataObjWrite(*D_FD,*,decodeBufInOut2);
        msiDataObjClose(*D_FD,*STAT_2);
    }
    msiOrbClose(*orbId);
    msiDataObjClose(*D_FD,*STAT_2);
}

openForAppendOrCreate(*SFile,*Resc,*D_FD,*New) {
    *SObj = "objPath=" ++ *SFile ++ "++++openFlags=O_RDWR"
    msiDataObjOpen(*SObj,*D_FD);
    msiDataObjLseek(*D_FD,*Offset,*Loc,*Status1);
    msiDataObjWrite(*D_FD,"\n",*WR_LN_F3);
    *New = 0;
}

openForAppendOrCreate(*SFile,*Resc,*D_FD,*New) {
    msiDataObjCreate(*SFile,*Resc,*D_FD);
    msiDataObjWrite(*D_FD,netcdf seis_waveform {ntypes:\n",*WR_LN_F);
msiDataObjWrite(*D_FD," double timestamp;\n",*WR_LN_F);
msiDataObjWrite(*D_FD," float upward;\n",*WR_LN_F);
msiDataObjWrite(*D_FD," float eastward;\n",*WR_LN_F);
msiDataObjWrite(*D_FD," float northward;\n",*WR_LN_F);
msiDataObjWrite(*D_FD," ); // seismic_vector_t\n",*WR_LN_F);
msiDataObjWrite(*D_FD, "dimensions:\n",*WR_LN_F);
msiDataObjWrite(*D_FD, "time = UNLIMITED;\n",*WR_LN_F);
msiDataObjWrite(*D_FD, "variables:\n",*WR_LN_F);
msiDataObjWrite(*D_FD, "seismic(time);\n",*WR_LN_F);
msiDataObjWrite(*D_FD, "seismic:standard_name = "three vector seismic data"\n;\n",*WR_LN_F);
msiDataObjWrite(*D_FD, "seismic:long_name = "Seismic";\n",*WR_LN_F);
msiDataObjWrite(*D_FD,"// global attributes:\n",*WR_LN_F);
*New = 1;
}

input *Coll="/rajaaf/home/rods/newenstest",
*Resc="destRescName=anfdemoResc++++forceFlag=",*Sensor="TA_J01E/MGENC/SM100",
*orbHost="anfexport.ucsd.edu:cascadia",*orbParam="",*modelIn=100,*Offset="-6",*Loc="SEEK_END",
*PKTNum=100000
output *pktId,*srcName,*oTime,*nBytes,*pktOut,*decodeBufInOut,ruleExecOut

7.5 Event log for collection formation (Policy 54)

Errors may occur in the sensor data as they are being generated (missing values or bad calibration), when the sensor data are archived (transmission error), and after storage (data corruption). Detection of errors on generation requires analysis of the data stream, test for values out of range, and tests for missing values. Detection of transmission errors can be handled with network protocols. Detection of errors after storage requires periodic validation of checksums. The following rule verifies the checksums of all files in the account /Mauna/home/atmos. Since the size of the collection is small, the rule does not need to monitor the load on the system. A log file is created that contains a time stamp for when the check was run, and that lists all corrupted files. The rule uses the policy functions:

\begin{verbatim}
checkCollInput
checkRescInput
createLogFile
findZoneHostName
isColl
\end{verbatim}

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-validate-chksum.r.

validateChecksums {
# dmp-validate.chksum.r
# Each file is checked to verify whether they have valid checksums
# Writes a log file stored as Check-Timestamp in directory *Coll/log
checkRescInput (*uncResc, $rodsZoneClient);
*NumBadFiles = 0;
*NumFiles = 0;
#================= check whether a collection was defined =================
checkCollInput (*Coll);
Data management policy: Collection reports (Policy 41)

Information about the collection may include the number of files, the size of the data, the number of metadata values, the usage, when integrity checks were done, the uniformity of metadata across the files, the size distribution, etc. The information may be organized by each sub-collection, or by file type, or by year. Reports are generated by issuing queries to the iCAT catalog and formatting the results. This example policy lists the size of each collection and the number of files that are publicly accessible. The rule uses the policy function:

- checkCollInput
- checkRescInput
- createLogFile
- findZoneHostName
- isColl

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-report.r.
When processing observational data, communities generate three additional classes of data: 1) calibrated data, 2) physical variables, and 3) gridded data. The processing steps can be aggregated into a processing pipeline that automatically generates each successive data class. The processing can be applied each time a file is deposited into a known directory, or applied in a batch mode at a remote compute server, or applied at the storage resource. The processing steps can also be captured in a workflow that is registered into the data grid. Each execution of the workflow can be tracked, associating the workflow input with the workflow output.

The following rule illustrates processing that is automatically applied each time a file is deposited into a specified collection. In this case a report is amended to add information about each file that is deposited.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acPostProcForPut-report.re.
7.8 Data category management (Policy 78)

The categories of data include observational, experimental, simulation, survey, and publications. Different assertions can be made about each type of data. Thus observational data needs to be calibrated, converted to physical variables, and mapped to a coordinate system. Experimental data may require additional provenance information that record the details of each experiment. Simulation data need close tracking of simulation version and input files. Publication data may have a release date that depends upon acceptance by a journal. In each case, a set of assertions are made about the data collection which are uniformly applied to all deposited files.

Similarly to the Product Generation task, data category management can be expressed as a set of processing steps that enforce the assertions. An example policy is the automated application of a processing step on the storage system holding the data. This rule executes an application (called app) stored in the irods/server/bin/cmd directory. Two input arguments are set up for the app, and the temporary files are deleted. The rule uses the policy function:

```r
checkPathInput
```

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-external-process.r.

```r
externalProcess {
# dmp-external-process.r
# Command to be executed is located in directory irods/server/bin/cmd
# This rule invokes an application that runs at the remote storage location
checkPathInput (*outXmlFile);
checkPathInput (*Pathf);
*timeStamp = double (time());
# Query the metadata catalog to get the Data ID for the input path, *Pathf
msiSplitPath(*Pathf, *Coll, *File);
# Now Make a query to get the absolute path to the file and the resource name
*Q = select DATA_PATH, DATA_RESC_NAME where DATA_NAME = *File’ and COLL_NAME = *Coll’;
foreach (*row in *Q) {
    *Path = *row.DATA_PATH;
    *Resource = *row.DATA_RESC_NAME;
}
# Make another query for IP Address of the resource
*Query2 = select RESC_LOC where DATA_RESC_NAME = *Resource’;
foreach (*row in *Query2) {*Addr = *row.RESC_LOC;}
# set up arguments for the command execution
*Arg1 = execCmdArg("-f");
*Arg2 = execCmdArg("-X");
if (errorcode(msiExecCmd(*Cmd,"*Arg1 *Arg2", "null", "*Pathf", "null", *Result)) < 0) {
    if(errormsg(*Result, *msg) == 0) {
        msiGetStderrInExecCmdOut(*Result,*Out);
        writeLine("stdout", "ERROR: *Out");
    } else {
        writeLine("stdout", "Result msg is empty");
    }
}
}
```
7.9 Re-using existing data (Policy 79)
A data grid can access files from external repositories. A local copy can be made and used in processing steps. Most repositories provide web services for accessing files. This example rule retrieves a file from a specified URL and stores a copy of the file in the data grid.

Updates to this policy are available from

7.10 Quality control (Policy 80)
Assertions about properties of a collection can be verified by periodically evaluating assessment criteria. The types of properties that can be verified include required metadata, required file type, integrity, distribution, etc.

The example rule compares the metadata defined on a collection and checks that each file in the collection has had the same metadata attributes defined. The rule uses the policy function:
checkCollInput

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-metadata-check-coll.r.

metadataCheck {
    # dmp-metadata-check-coll.r
    # compare the metadata on each file with the metadata on the collection
    # retrieve the metadata on the collection
    checkCollInput (*Coll);
    *Q = select META_COLL_ATTR_NAME where COLL_NAME = '*Coll';
    foreach (*R in *Q) {
        *MetaN = *R.META_COLL_ATTR_NAME;
        # check that each file also has this metadata attribute
        *Q1 = select DATA_ID, DATA_NAME where COLL_NAME = '*Coll';
        foreach (*R1 in *Q1) {
            *Datald = *R1.DATA_ID;
            *Q3 = select count(META_DATA_ATTR_ID) where DATA_ID = '*Datald' and META_DATA_ATTR_NAME = '*MetaN';
            foreach (*R3 in *Q3) {*Num = *R3.META_DATA_ATTR_ID;}
            if (*Num == "0") {
                *Name = *R1.DATA_NAME;
                writeLine ("stdout", "*Name missing "MetaN");
            }
        }
    }
}

INPUT *Coll = "/dfcmain/home/rwmoore/Reports"
OUTPUT ruleExecOut

7.11 Analysis procedures (Policy 81)
Each time a file is added to the system, a new file version is created. A version of a file can be created by adding a time stamp, and moving the version to an archive directory. This rule processes files in a collection, creating a version of each file that is stored in a destination directory called “SaveVersions”. The rule is called ruleversion.r and is listed in section 4.7.1.

The version number can be inserted in the file name before the extension. This rule parses the file name, identifies an extension, and inserts the time stamp before the extension when the version name is created. The rule is automatically executed within the acPostProcForPut policy enforcement point. Note that access controls have to be set on the versioned file. The rule is called ruleversionfile.r and is listed in section 4.7.1.

The rule “ruleversionfile.r” can be modified to enforce versioning at a Policy Enforcement Point. The following rule is applied every time a file is loaded into the data grid.

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/acPostProcForPut-version.re.
acPostProcForPut {
    # acPostProcForPut-version.re
    # create a copy of the file by modifying the file name with a version stamp
    # note that a collection must be specified where the version will be stored
    *Path = $objPath;
    msiSplitPath(*Path, *Coll, *File);
    # construct version name
    msiGetSystemTime(*Tim, "human");
    # check whether there is a file extension on the name
    *Fstart = *File;
    *Fend = ";
    *out = errormsg(msiSplitPathByKey (*File, ",", *Fstart, *Fend, *Msg);
    *Vers = *Fstart ++ "," ++ *Tim ++ *Fend;
    *Pathver = "/$rodsZoneClient/home/$userNameClient/version/" ++ *Coll ++ "/" ++ *Vers;
    msiDataObjCopy(*Path, *Pathver, "forceFlag=[,Status);
    msiSetACL("default", "own", $userNameClient, *Pathver);
}

7.12 Analysis collaborations (Policy 82)
When collaborations result in multiple persons updating a collection, a change log
will be needed to determine when updates have been made to a collection. Two
approaches are to analyze audit trails, or to periodically summarize the contents
of the collection.

A change log summarizes all changes made to the sensor data. The change log can
be created by listing all of the files that are in the "/Mauna/home/atmos/version"
directory. The rule uses the policy function

    checkRescInput
    createLogFile
    findZoneHostName
    isColl

Updates to this policy are available from

ListChanges {
    # dmp-report-changes.r
    # List all files in the version directory
    #============= create a collection for log files if it does not exist ==============
    checkRescInput (*Res, $rodsZoneClient);

    *Q2 = select DATA_NAME, COLL_NAME where COLL_NAME like
    "/Mauna/home/atmos/version%";
    foreach (*R2 in *Q2) {
        *Coll = *R2.COLL_NAME;
        *File = *R2.DATA_NAME;
        writeLine("Lfile", "*Coll/*File");
    }
}
A reserved vocabulary can be implemented for a collection using the HIVE (Helping Interdisciplinary Vocabulary Engineering) system. HIVE maintains an ontology for a discipline, defining relationships between words as well as a standard vocabulary. The descriptive metadata registered on files within a collection can be checked for compliance with the reserved vocabulary. This ensures that well-known terms can be used to query the collection and identify relevant material.

An example validation rule utilizes a REST service to iterate over iRODS collections, validating the terms as being valid SKOS references, and generating a report on invalid terms. The rule is called validate-ontologies.r and is listed in section 5.12.1.

An example output for when two data objects are annotated, one with an invalid term, is listed below.

test1@ubuntu:~workspace/rule_workbench$ irule -F validate_data_object_ontologies.r
Metadata validation report
/fedZone1/home/rods/hive/libmsiCurlGetObj.cpp has uri http://purl.org/astronomy/uat#TT888 that is not in a valid ontology

The ingestion of data into the collection is governed by processes outside of iRODS. If an Antelope Real Time System is being used to manage the sensor data, then micro-services exist to automate the periodic ingestion of sensor records from ARTS into an iRODS collection. The update can be done periodically. Note that the attribute DATA_CREATE_TIME is automatically set each time a file is created, and DATA_MODIFY_TIME is automatically set each time a file is modified. The rule is called dmp-sensor-harvest.r and is listed in section 7.4.

A check can be made that the data type associated with each sensor data file is .csv. The rule uses the policy function:

```
checkCollInput
```

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-checkDataType.r.

```r
checkDataType {
  # dmp-checkDataType.r
  # Verify that all files within the sensor data collection have extension .csv
  # except for the .xml files
  checkCollInput (*Col);  
  *Q1 = select DATA_NAME, DATA_TYPE_NAME, COLL_NAME where COLL_NAME like '*Col%';
  foreach (*R1 in *Q1) {
    *Type = *R1.DATA_TYPE_NAME;
```
if (*Type != "csv" && *Type != "xml") {
    *File = *R1.DATA_NAME;
    *Coll = *R1.COLL_NAME;
    writeLine("stdout", "Found an invalid data type *Type for *Coll/*File");
}

INPUT *Col = "/Mauna/home/atmos/sensor"
OUTPUT ruleExecOut

7.16 Unique identifiers (Policy 27)
A Digital Object Identifier can be generated automatically through an extension to
the acPostProcForPut rule. The Handle system can use a local handle registry for
assigning identifiers to files. The local handle registry, in turn, is assigned a unique
identifier in a global handle system.

The following rule creates a handle and registers it in the DFC handle server:
(the registration of the handle in our handle server indicates it is available for access
from DataONE.)

Updates to this policy are available from

    # rules for creating handle based unique identifiers
    # just creating handles for nexrad data right now

acPostProcForPut {
    # Attempting to create Handle for $objPath
    *Cmd = "create_handle";
    *Keyfile = "/var/lib/irods/hs/admpriv.bin";
    *Uri = "irods%3A%2F%2Firen2.renci.org%3A1237$objPath";
    *Url = "https://dfcweb.datafed.org/idrop-web2/home/link?irodsURI=*Uri";

    *Args = "$dataId $Url";
    msiExecCmd(*Cmd, *Args, "null", "null", "null", *Result);
    msiGetStdoutInExecCmdOut(*Result,*Out);
    # Created Handle *Out for $objPath
}

The rule executes a shell script:

    #!/bin/bash

    if [ "$#" -ne 2 ]; then
        echo "Usage: create_handle <data object id> <data object url>"
        exit 1
    fi

    OID="$1"
    URL="$2"

    HANDLE=$(java -classpath ./irods-hs-tools.jar org.irods.dfc.CreateHandle
The metadata attributes that will be created can be specified in a template. Depending upon the sensor data format, the attributes can be parsed from each sensor file and added as metadata on the file. Examples exist for parsing metadata from text files, netCDF files, XML files, etc. Pattern matching operations can be applied to text to extract contextual metadata. A template for pattern matching can be created that defines triplets:

\[\text{<pre-string-regexp, keyword, post-string-regexp>}.\]

The triplets are read into memory, and then used to search a data buffer. For each set of pre and post regular expressions, the string between them is associated with the specified keyword and can be stored as a metadata attribute on the file.

In the example, the template file has the format:

\[
\text{<PRETAG>X-Mailer: </PRETAG>Mailer User</POSTTAG>}
\text{<PRETAG>Date: </PRETAG>Sent Date</POSTTAG>}
\text{<PRETAG>From: </PRETAG>Sender</POSTTAG>}
\text{<PRETAG>To: </PRETAG>Primary Recipient</POSTTAG>}
\text{<PRETAG>Cc: </PRETAG>Other Recipient</POSTTAG>}
\text{<PRETAG>Subject: </PRETAG>Subject</POSTTAG>}
\text{<PRETAG>Content-Type: </PRETAG>Content Type</POSTTAG>}
\]

The end tag is actually a "return" for unix systems, or a "carriage-return/line feed" for Windows systems. The example rule reads a text file into a buffer in memory, reads in the template file that defines the regular expressions, and then parses the text in the buffer to identify presence of a desired metadata attribute. The rule is called rulemetaload.r and is listed in section 4.6.3.

7.18 Metadata export (Policy 84)

The descriptive metadata that are registered on each file can be extracted and written as an XML file. This rule creates an XML metadata file for each file in the /Mauna/home/atmos/sensor directory. The following structure is used:

\[
\text{<?xml version="1.0"?>}
\text{<catalog>}
\text{<File path="COLL_NAME/DATA_NAME">}
\text{<META_DATA_ATTR_NAME>META_DATA_ATTR_VALUE</META_DATA_ATTR_NAME>}
\text{</File>}
\text{</catalog>}
\]

183
The name of the metadata file is created by appending .xml to the name of the sensor data file. The rule uses the policy functions:

- checkCollInput
- checkRescInput
- findZoneHostName

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-createXML.r.

```r
createXML {
  # dmp-createXML.r
  # for each file in a collection
  # create an associated XML file
  *Xhead =``<?xml version="1.0"?>``;
  *Xcat =``<catalog>``;
  *Xend =``</catalog>``;
  *Coll = "/$rodsZoneClient/home/$userNameClient/*Relcoll";
  checkCollInput (*Coll);
  checkRescInput (*Res, $rodsZoneClient);
  *Q1 = select COLL_NAME, DATA_NAME where COLL_NAME like "*Coll%";
  foreach (*R1 in *Q1) {
    *File = *R1.DATA_NAME;
    *Col = *R1.COLL_NAME;
    *Filn = *File ++ ".xml";

    # =========== create an XML file if it does not exist ===========

    *Lfile = "*Col/*Filn";
    *Dfile = "destRescName=*Res++++forceFlag=";
    msiDataObjCreate(*Lfile, *Dfile, *L_FD);
    writeLine("*Lfile","*Xhead");
    writeLine("*Lfile","*Xcat");
    writeLine("*Lfile","<File path="*Col/*File">");
    *Q2 = select META_DATA_ATTR_NAME, META_DATA_ATTR_VALUE where DATA_NAME = 'File' and COLL_NAME = 'Col';
    foreach (*R2 in *Q2) {
      *Aname = *R2.META_DATA_ATTR_NAME;
      *Aval = *R2.META_DATA_ATTR_VALUE;
      writeLine("*Lfile","<"*Aname">"*Aval"</"*Aname">");
    }
    writeLine("*Lfile","</File>");
    writeLine("*Lfile","*Xend");
    msiDataObjClose(*L_FD, *Status);
  }
}
```

**INPUT** *Relcoll ="sub1", *Res =$"demoResc"

**OUTPUT** ruleExecOut

### 7.19 Collection creation system (Policy 85)

The data management plan should include information about the system that will be used to assemble the collection. This may be different from the system used to
archive the collection. A collaboration environment facilitates collection creation. Each collaborating person is given an account, and permissions are set to allow deposition of files into the shared collection. This requires:

- Creating shared collection name. This may be a separate account in the data grid.
- Setting write access controls on the shared collection. This may be done by creating a user group that is allowed to update the collection.
- Defining the desired naming convention for the files. This may require renaming each file as it is deposited.
- Defining the required provenance and descriptive metadata needed for each file. This may require extraction of header information from each file.

The following policy lists the names of the persons in each group that can update the shared collection. The rule uses the policy function:

checkCollInput

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-check-group.r.

checkGroup {
  # dmp-check-group.r
  # List the persons that can access a collection
  checkCollInput (*Coll);
  writeLine("stdout", "Collection *Coll can be manipulated by");
  *Q1 = select COLL_ID where COLL_NAME = '*Coll';
  foreach (*R1 in *Q1) {*CollID = *R1.COLL_ID;}
  *Q2 = select COLL_ACCESS_USER_ID, COLL_ACCESS_TYPE where COLL_ACCESS_COLL_ID = '*CollID';
  foreach (*R2 in *Q2) {
    *UserID = *R2.COLL_ACCESS_USER_ID;
    *Type = *R2.COLL_ACCESS_TYPE;
    *Q5 = select TOKEN_NAME where TOKEN_NAMESPACE = 'access_type' and TOKEN_ID = *Type;
    foreach (*R5 in *Q5) {*Access = *R5.TOKEN_NAME;}
    if (*Access == 'own' || *Access == 'create object') {
      *Q3 = select USER_NAME where USER_ID = '*UserID';
      foreach (*R3 in *Q3) {
        *Usr = *R3.USER_NAME;
        writeLine("stdout", *Usr);
      }
      *Q6 = select count(USER_NAME) where USER_GROUP_ID = '*UserID';
      foreach (*R6 in *Q6) {
        *Nsg = *R6.USER_NAME;
        if(int(*Nsg) > 1) {
          *Q7 = select USER_NAME where USER_GROUP_ID = '*UserID';
          foreach (*R7 in *Q7) {
            *Usg = *R7.USER_NAME;
            writeLine("stdout", *Usg);
          }
        }
      }
    }
  }
}
The total size of the collection can be found by querying the iCAT catalog. The total size should include the storage space for replicas, the storage space for intermediate products, and the storage space for published results.

The example policy takes as input a collection name. The rule uses the policy functions:

- checkCollInput
- checkRescInput
- createLogFile
- findZoneHostName
- isColl

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-report-size.r.

A standard approach is to place the restricted access data in a collection, create user groups for allowed users, and restrict access to just the allowed user groups.
There are three types of data managed by the Mauna Loa project: sensor data, derived data products, and research data. These can be handled by creating three collections:

- /Mauna/home/atmos/sensor
- /Mauna/home/atmos/derived
- /Mauna/home/atmos/research

We will turn on inheritance in each collection, and set the access controls at the collection level.

Public access is specified for all sensor data for the Mauna Loa data. In the iRODS data grid, public access is through the “anonymous” account. We turn on inheritance on the “sensor” data collection and give access to the “anonymous” account. The rule uses the policy function:

checkCollInput

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-set-public.r.

```
myTestRule {
    # dmp-set-public.r
    # Input parameters are:
    #  Recursion flag
    #    default
    #    recursive - valid if access level is set to inherit
    #  Access Level
    #    null
    #    read
    #    write
    #    own
    #    inherit
    #  User name or group name who will have ACL changed
    #  Path or file that will have ACL changed
    *Home="/\$rodsZoneClient/home/\$userNameClient/";
    *Path=*Home ++ *RelativeCollection;
    checkCollInput (*Path);
    msiSetACL("recursive", "inherit", "anonymous",*Path);
    msiSetACL("default", "read", "anonymous", *Path);
    writeLine("stdout", "Set inheritance on *Path and access to anonymous");
}
```

INPUT  *RelativeCollection="sensor"
OUTPUT ruleExecOut

### 7.22 Publication of data products (Policy 88)

The time periods for holding data proprietary varied across the DMPs, and examples included 6 months, 2 years, until project end, until project review, and until research publication. For the Mauna Loa data, all derived data will be held private until a six month period has elapsed. At the end of this period we change the read access to public. The rule uses the policy function:

checkCollInput
Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-proprietary-change.r.

```r
myTestRule {
# dmp-proprietary-change.r
# check files for expiration of proprietary period (6 months from creation)
# and set public access
*Home="/rodsZoneClient/home/$userNameClient/";
*Path=*Home ++ *RelativeCollection;
checkCollInput (*Path);
msiGetSystemTime(*TimeU,"unix");
*Del=str(182 * 24 * 3600);
# Loop over files
Q1 = select DATA_NAME, DATA_CREATE_TIME, DATA_ID where COLL_NAME = 'Path';
foreach (*R1 in *Q1) {
*File=*R1.DATA_NAME;
*Create=*R1.DATA_CREATE_TIME;
*Dataid=*R1.DATA_ID;
# Check for files past proprietary period
if (*TimeU - Create >= *Del) {
*Query4 = select DATA_ACCESS_TYPE, DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = *Dataid';
# Loop over access controls for each file to avoid replicating access controls
*Count = 0;
foreach(*Row4 in *Query4) {
*Userdid=*Row4.DATA_ACCESS_USER_ID;
*Datatype=*Row4.DATA_ACCESS_TYPE;
if(*Userdid == "anonymous" && *Datatype == "read") {
*Count = 1;
}
}
if(*Count == 0) {
msiSetACL("default","read","anonymous",*Path);
writeLine("stdout","Set public access on *Path");
}
}
}
```

INPUT *RelativeCollection="derived", *Ad = "read"
OUTPUT ruleExecOut

### 7.23 Re-use policies (Policy 89)

Collection re-use occurs when the collection is subsumed into another digital library, or processed through a new data processing pipeline, or archived at another site. Depending upon the type of data, re-use may entail multiple requirements:

- **Access permission.** All proprietary or confidential data require negotiation of access agreements. This may require anonymization of data files, or encryption of data files, or creation of access controls.
- **Descriptive metadata.** The context associated with each file is represented by a standard metadata schema. Re-use may require mapping from the chosen standard to another metadata schema. The HIVE technology provides the ability to map between ontologies to simplify this process.
- **Integrity checks.** Integrity should be verified on each shared data object. This implies the community that is re-using the data can verify checksums on each file.

- **Policy-encoded objects.** The policies that govern access and processing of a digital object can be encapsulated with the digital object. If these policies are automatically loaded into a controlling rule engine when the digital object is used, control can be maintained even when the digital object is re-used. The implementation will require:
  - Encryption of the digital object.
  - Negotiation between the institution that is re-using the digital object and the original repository for the encryption key.
  - Verification that the re-use institution is capable of enforcing the policies.
  - Extraction of the associated policies and there loading into a re-use rule engine.

- **Preservation of Digital Object Identifiers.** The metadata used to identify the digital objects should be preserved by the re-use institution.

- **Provenance trail.** Digital objects that are derived from the original data should include metadata that denotes the source and the transformation that were applied to the original data. The transformations can be encapsulated in workflows that can be registered into the repository along with identifiers for the input files and the output files.

The implementation of these policies depends upon the technology used by the re-use institution. If data grid technology is used, many of these requirements may be implemented through federation of the original data grid and the re-use data grid.

### 7.24 Distribution policies (Policy 90)

Researchers prefer to have a local copy of the data sets they are analyzing. This minimizes latency in processing pipelines, ensures access, and enables tracking of versions of the data without disrupting the original collection. Distribution policies may be defined to:

- Cache data on a resource at a remote institution.
- Control which data sets may be re-used.
- Automate generation of copies at the remote site when files are added to a collection.
- Distribute files across institutions depending upon the type of data. An example is the distribution of sensor data to the institution that is working with a particular sensor.
- Apply transformative migration as the data sets are distributed to ensure the appropriate format is provided.
- Distribute workflows that can be used to process the data sets.
- Distribute applications within Docker virtual environment images that can be used to analyze the data sets.
- Distribute the descriptive metadata either as an XML file, or a CSV file, or a JSON file.

The following policy generates a JSON file containing the descriptive metadata for the data files in a collection. For each file, a JSON file is put into a subdirectory called "Metadata". The rule uses the policy functions:
- checkCollInput
- checkRescInput
- findZoneHostName
- isColl

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-json.r.

```r
jsonFileMeta {
# dmp-json.r
# Create a json file of all descriptive metadata for a file
# Write files into a separate metadata collection
#=================== create a collection for reports if it does not exist ==================
checkCollInput (*Coll);
checkRescInput (*Res, $rodsZoneClient);
*LPath = "*Coll/Metadata";
isColl(*LPath, "stdout", *Status);
if (*Status >= 0) {
*Q1 = select DATA_NAME where COLL_NAME = "*Coll";
foreach (*R1 in *Q1) {
 *File = *R1.DATA_NAME;
#=================== create file into which results will be written ===================
*Lfile = "*LPath/" ++ "$File" ++ ".json";
*Dfile = "destRescName=*Res++++forceFlag=";
msiDataObjCreate(*Lfile, *Dfile, *L_FD);
writeLine("*Lfile", \{);
writeLine("*Lfile", "DATA_NAME": "*File",");
*Q3 = select count(META_DATA_ATTR_NAME) where DATA_NAME = "*File" and COLL_NAME = "*Coll";
foreach (*R3 in *Q3) {*Num = int(*R3.META_DATA_ATTR_NAME);}
*Count = 0;
*Q2 = select META_DATA_ATTR_NAME, META_DATA_ATTR_VALUE, META_DATA_ATTR_UNITS where DATA_NAME = "*File" and COLL_NAME = "*Coll";
foreach (*R2 in *Q2) {
 *Name = *R2.META_DATA_ATTR_NAME;
 *Value = *R2.META_DATA_ATTR_VALUE;
 *Units = *R2.META_DATA_ATTR_UNITS;
 *Count = *Count + 1;
 writeLine("*Lfile", \{");
writeLine("*Lfile", "META_DATA_ATTR_NAME": "*Name",");
writeLine("*Lfile", "META_DATA_ATTR_VALUE": "*Value",");
writeLine("*Lfile", "META_DATA_ATTR_UNITS": "*Units",");
if (*Count == *Num) {writeLine("*Lfile", \})
else {writeLine("*Lfile", ")");
}
}
```
There are no restrictions on access for the Mauna Loa sensor data. Typical access restrictions for other DMPs include Institutional Review Board, proprietary data, and copyright. As before, the restrictions can be enforced by placing restricted data in a collection, creating user groups for the allowed users, and only permitting allowed groups to access the data. A standard task is to verify that the access controls have been set correctly. The rule uses the policy functions:

```r
checkUserInput
contains
findZoneHostName
```

Updates to this policy are available from

http://github.com/DICE-UNC/policy-workbook/dmp-group-access.r.

```r
checkAccess {
  # dmp-group-access.r
  # verify that access is restricted to a designated group of users
  # Get USER_ID for the input group name
  checkUserInput (*Group, $rodsZoneClient);
  *Query = select USER_ID where USER_NAME = '*Group';
  *Userid = "";
  foreach(*Row in *Query) {
    *Userid = *Row.USER_ID;
  }
  if(*Userid == "") {
    writeLine("stdout","Input group name *Group is unknown");
    fail;
  } else {writeLine("stdout","Group ID is *Userid");}
  # loop over files in home collection
  *Coll = "/$rodsZoneClient/home/$userNameClient%";
  *rs = select DATA_ID, DATA_SIZE where COLL_NAME like '*Coll';
  *res.total = str(0);
  *total.total = str(0);
  foreach(*r in *rs) {
    *fn = *.r.DATA_ID;
    *ds = *.r.DATA_SIZE;
    # Find ACL for the file
    *Query4 = select DATA_ACCESS_TYPE, DATA_ACCESS_USER_ID where DATA_ACCESS_DATA_ID = '*fn';
    # Loop over access controls for each file
    foreach(*Row4 in *Query4) {
      # Code continues here...
    }
  }
}
```
*Userdid = *Row4.DATA_ACCESS_USER_ID;
*Datatype = *Row4.DATA_ACCESS_TYPE;
if(*Userid != *Userdid) {
    *Query5 = select TOKEN_NAME where TOKEN_NAMESPACE = 'access_type' and TOKEN_ID = *
    *Datatype;
    foreach (*Row5 in *Query5) {*Access = *Row5.TOKEN_NAME;}
    *Query6 = select USER_NAME where USER_ID = *Userid';
    foreach (*Row6 in *Query6) {*Usern = *Row6.USER_NAME;}
    *res.*Usern = str(int(*res.*Usern) + 1)
    *total.*Usern = str(double(*res.*Usern) + double(*ds))
} else {
    *res.*Usern = str(1);
    *total.*Usern = *ds;
}

*res.total = str(int(*res.total) + 1);  
*total.total = str(double(*total.total) + double(*ds));
}
writeLine("stdout", "usern	count	avg	size
total size");
foreach(*Usern in *res) {
    *Us = "*Usern\t";
    if(strlen(*Usern) >= 8) {*Us = "*Usern\t";
    if("*Usern != "total") {
        writeLine("stdout",
        "*Us"++*res.*Usern++"\t"++str(double(*total.*Usern)/int("*res.*Usern))++"\t"++*total.*Us
        ern);
    }
    writeLine("stdout",
    "total\t"++*res.total++"\t"++str(double(*total.total)/int(*res.total))++"\t"++*total.total)
;}

INPUT *Group ="engineering"
OUTPUT ruleExecOut

7.26 IPR restrictions (Policy 91)
We assume that files deposited into the research directory have been published. To ensure public access, we only need to set inheritance on the directory for the “anonymous” account. This can be done as shown for Task 1. This rule uses the policy function:
checkCollInput

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/odum-inherit.r.

myTestRule {
    # odum-inherit.r
    # Input parameters are:
    # Recursion flag
A more sophisticated rule would check for a metadata flag that specifies that publication has been done. This rule checks whether the value of a “PUBLICATION” flag is set to 1, and then provides public access. The rule uses the policy functions:

- addAVUMetadata
- checkCollInput
- deleteAVUMetadata

Updates to this policy are available from http://github.com/DICE-UNC/policy-workbook/dmp-publication.r.
if(*Name == "PUBLICATION") {
    *Count = *Count + 1;
    if(*Val != "1") {
        # remove old PUBLICATION flag
        deleteAVUMetadata (*Path, "PUBLICATION", *Val, ",", *Status);
        # Set PUBLICATION flag to 1
        if (*Count == 1) {
            addAVUMetadata (*Path, "PUBLICATION", "1", "", *Status);
            writeLine("stdout", "Set PUBLICATION for *Coll1/*File");
        }
    }
}
if (*Count == 0) {
    # Set PUBLICATION flag to 1
    addAVUMetadata (*Path, "PUBLICATION", "1", "", *Status);
    writeLine("stdout", "Set PUBLICATION for *Coll1/*File");
}
# Check for public access on file
*Q3 = select count(DATA_ACCESS_USER_ID) where DATA_ACCESS_DATA_ID = '*FileId' and DATA_ACCESS_USER_ID = '*UserIdPublic';
foreach (*R3 in *Q3) {
    *Count = *R3.DATA_ACCESS_USER_ID;
    if (*Count == '0') {
        *File = *R1.DATA_NAME;
        *Path = "*Coll1/*File";
        msiSetACL("default", "read", "public", *Path);
        writeLine("stdout", "Enabled access for public in *File in *Coll1");
    }
}
*Q4 = select count(DATA_ACCESS_USER_ID) where DATA_ACCESS_DATA_ID = '*FileId' and DATA_ACCESS_USER_ID = '*UserIdAnon';
foreach (*R4 in *Q4) {
    *Count = *R4.DATA_ACCESS_USER_ID;
    if (*Count == '0') {
        *File = *R1.DATA_NAME;
        *Path = "*Coll1/*File";
        msiSetACL("default", "read", "anonymous", *Path);
        writeLine("stdout", "Enabled access for anonymous in *File in *Coll1");
    }
}
INPUT *Coll = "/Mauna/home/atmos/research"
OUTPUT ruleExecOut

7.27 Web access policies (Policy 92)
A standard approach across the DMPs is to provide a persistent URL for accessing data sets. Within the iRODS data grid, either a URL can be created for public access, or a ticket can be created that defines a persistent URL, defines access controls, and also defines the time period over which the ticket is valid. Any person holding the ticket is allowed access to the data set. Tickets can be created by a web client, or can
be created by running the `iticket` command. A rule can be created to list tickets used within a collection. The rule uses the policy function:

```
checkCollInput
```

Updates to this policy are available from

http://github.com/DICE-UNC/policy-workbook/dmp-list-tickets.r.

```r
checkTicket {
  # dmp-list-tickets.r
  # list the tickets created for a collection
  checkCollInput (*Coll);
  *rs = select COLL_NAME where COLL_NAME like '*Coll%';
  foreach(*r in *rs) {
    *Col = *r.COLL_NAME;
    # Find tickets for the file
    *Query4 = select TICKET_ID, TICKET_EXPIRY where TICKET_DATA_COLL_NAME = '*Col';
    # List tickets for each collection
    foreach(*Row4 in *Query4) {
      *Tid = *Row4.TICKET_ID;
      *Texp = *Row4.TICKET_EXPIRY;
      writeLine("stdout","Ticket *Tid expires *Texp");
    }
  }
}
```

**INPUT** *Coll = "$\$/Mauna/home/atmos/research"

**OUTPUT** ruleExecOut

### 7.28 Data sharing system (Policy 93)

The choice of the data management system for sharing or publishing the data products depends on the type of data product. Most DMPs use GitHub to publish code, a database to publish information, and a data repository to publish data. In each of these cases, the data sets are typically publicly accessed. For finer grain access control, a digital repository or data grid is chosen. The data sharing system should provide the following capabilities:

- **Collection hierarchy.** This is needed to separate the generation of data from the publication of data.
- **Access controls.** Usually intermediate data products are not released to the public. Derived data products are usually held proprietary until they are verified for quality.
- **Support for distributed data.** Data products may be located at multiple sites and should be managed by the data sharing system.

### 7.29 Code distribution system (Policy 94)

The distribution of code may be done through an open source code repository such as GitHub, or through a web site, or even through a data repository. The major challenges are the management of versions, the development of documentation, and unit testing to verify all updates.
7.30 Retention period (Policy 21)
The retention period for the data products is usually measured in years. A challenge, then, is how to show that the data products were retained for the required length of time. One approach is to turn off deletion on the data collection. This rule can be updated in the iRODS core.re file.

The rule is available at
https://github.com/DICE-UNC/policy-workbook/blob/master/acDataDeletePolicy-collection.re

```plaintext
acDataDeletePolicy {ON($objPath like "/$rodsZoneClient/home/$userNameClient/sensor/*") {
  msiDeleteDisallowed;
}
}
```

This prohibits deletion even by an administrator. The files in the collection can then be checked for whether their retention period has been passed. The rule uses the policy function:

```plaintext
checkCollInput
```

Updates to this policy are available from

```plaintext
checkRetention {
  # dmp-check-retention.r
  # Identify files whose retention period has expired
  checkCollInput (*Coll);
  msiGetIcatTime("Time","unix");
  *Q1 = select DATA_NAME, COLL_NAME, DATA_EXPIRY where COLL_NAME like "*Coll%";
  foreach (*R1 in *Q1) {
    *D = *R1.DATA_NAME;
    *C = *R1.COLL_NAME;
    *E = *R1.DATA_EXPIRY;
    if (int("Time") > int("E")) {
      writeLine("stdout","*C/*D retention period has expired");
    }
  }
  INPUT *Coll =$"/Mauna/home/atmos/sensor"
  OUTPUT ruleExecOut
```

7.31 Curation plans (Policy 95)
Curation activities include:

- Validation of descriptive metadata
- Validation of provenance metadata
- Setting of access controls
- Verification of data formats

The curation policies can be registered into the iCAT catalog. The policies can then be retrieved from the catalog and published as a report. The example policy lists all of the policies that are being enforced at policy-enforcement points within the iRODS data grid. The rule uses the policy function:
checkCollInput
checkRescInput
createLogFile
findZoneHostName
isColl

Updates to this policy are available from

listRules {
  # dmp-pepRules.r
  # Generate a report listing the rules that are automatically enforced
  #==================== create a collection for reports if it does not exist ====================
  checkCollInput (*Coll);
  checkRescInput (*Res, $rodsZoneClient);

  msiAdmShowIRB(*A);
  msiDataObjWrite(*FD,"stdout",WLEN);
  msiDataObjClose(*FD,*Status);
}
INPUT *Coll = "/$rodsZoneClient/home/$userNameClient/Reports", *Res = "demoResc"
OUTPUT ruleExecOut

7.32 Archive system (Policy 96)
For long term storage, a deposition will be required into the remote archive. If two
data grids are federated, then a rule can be run to archive all files from a selected
collection into the remote storage location. The rule uses the policy functions:
checkCollInput
checkRescInput
createLogFile
findZoneHostName
isColl

Updates to this policy are available from

archiveRule {
  # dmp-archive.r
  # Loop over files in a collection, *Src
  # Copy all files into an archive collection. *Dest
  checkCollInput (*Src);
  checkCollInput (*Dest);
  checkRescInput (*Res, *DestZone);
  *Len = strlen(*Src);

  #====================get current time, Timestamp is YYYY-MM-DD.hh:mm:ss
  #================================
  msiGetSystemTime("TimeA", "unix");

  #==================== create a collection for log files if it does not exist ====================

# =========== find files to archive
*Query = select DATA_NAME, DATA_CHECKSUM, COLL_NAME where COLL_NAME like '%Src%';
foreach(*Row in *Query) {
*File = *Row.DATA_NAME;
*Check = *Row.DATA_CHECKSUM;
*Coll = *Row.COLL_NAME;
*L1 = strlen(*Coll);
*SrC1 = *Coll ++ "/" ++ *File;
*C1 = substr(*Coll,*Len,*L1);
if(strlen(*C1)==0) {
  *DestColl = *Dest;
  *Dest1 = *Dest ++ "/" ++ *File;
} else {
  *DestColl = *Dest ++ *C1;
  *Dest1 = *Dest ++ *C1 ++ "/" ++ *File;
}
isColl(*DestColl, *Lfile, *Status);
if (*Status >= 0) {
  msiDataObjCopy(*Src1,*Dest1,"destRescName=*Res++++forceFlag=", *Status);
  msiSetACL("default","own","*Acct", *Dest1);
  msiDataObjChksum(*Dest1, "forceChksum=",*Chksum);
  if (*Check != *Chksum) {
    writeLine("*Lfile", "Bad checksum for file *Dest1");
  } else { writeLine("*Lfile", "Moved file *Src1 to *Dest1");
    writeLine("*Lfile", "Moved file *Src1 to *Dest1");
  }
}
}

INPUT *Res=$"stage", *DestZone=$"ornlZone", *Acct=$"Mauna-acct",
*Src=$"/Mauna/home/atmos/sensor", *Dest =$"/*DestZone/home/*Acct#Mauna/archive"
OUTPUT ruleExecOut

7.33 Replication policy (Policy 13)
The number of replicas can be verified for each file in a collection. This rule lists all
files for which the required number of replicas is not available. The rule uses the
policy function:
  checkCollInput
  checkRescInput
  createLogFile
  findZoneHostName
  isColl

Updates to this policy are available from

    checkReplicas{
      # dmp-check-replicas.r

198
# Loop over all files in the specified collection
# ============= create a collection for log files if it does not exist =============
checkCollInput (*Coll);
checkRescInput (*Res, $rodsZoneClient);

*Query = select DATA_NAME,COLL_NAME where COLL_NAME like '*Coll%';
foreach(*Row in *Query){
    *Col = *Row.COLL_NAME;
    *Data = *Row.DATA_NAME;
    *temp_count=0;
    *Query2 = select count(DATA_ID) where COLL_NAME='*Col' and DATA_NAME='*Data';
    # For every coll/data find those which do not have the required number of replicas
    foreach(*Row2 in *Query2){
        *temp_count = *Row2.DATA_ID;
    }
    if(int(*temp_count) < *Numrep){
        *n = *Numrep - int(*temp_count);
        writeLine("*Lfile","*Col/*/Data is missing *n replicas");
    }
}

INPUT *Coll =$"/Mauna/home/atmos/sensor", *Res =$"maunaResc", *Numrep =$3
OUTPUT ruleExecOut

7.34 Backup policy (Policy 97)
The time period between backups can be set by specifying a periodic rule execution for archiving data. We can turn the rule specified for Task 18 into a periodic rule that is executed every 7 days. The rule uses the policy functions:
checkCollInput
checkRescInput
createLogFile
findZoneHostName
isColl
isData

Updates to this policy are available from
http://github.com/DICE‐UNC/policy‐workbook/dmp‐periodic‐backup.r.

    periodicArchive {
        # dmp‐periodic‐backup.r
        checkCollInput (*Src);
        checkCollInput (*Dest);
        checkRescInput (*Res, *DestZone);
        # periodically archive a collection every week
        delay("<PLUSET>1m</PLUSET><EF>7d</EF>") {
            collArchive (*Res, *Src, *Dest, *Acct);
        }
        writeLine("stdout","Periodic rule queued for archiving a collection");
    }
        # Loop over files in a collection, *Src
# Copy all files into an archive collection. *Dest
*Len = strlen(*Src);
### get current time, Timestamp is YYY-MM-DD:hh:mm:ss
msiGetSystemTime(*TimeA, "unix");
### create a collection for log files if it does not exist
### find files to archive
*Query = select DATA_NAME, DATA_CHECKSUM, COLL_NAME where COLL_NAME like "*Src%";
foreach(*Row in *Query) {
    *File = *Row.DATA_NAME;
    *Check = *Row.DATA_CHECKSUM;
    *Coll = *Row.COLL_NAME;
    *L1 = strlen(*Coll);
    *Src1 = *Coll ++ "/" ++ *File;
    *C1 = substr(*Coll,*Len,*L1);
    if(strlen(*C1)==0) {
        *DestColl = *Dest;
        *Dest1 = *Dest ++ "/" ++ *File;
    } else {
        *DestColl = *Dest ++ *C1;
        *Dest1 = *Dest ++ *C1 ++ "/" ++ *File;
    }
    isColl(*DestColl, *Lfile, *Status1);
    if (*Status1 == 0) {
        # If file is already in the archive, do not copy it again
        isData(*DestColl, *File, *Status);
        if (*Status == "0") {
            msiDataObjCopy(*Src1,*Dest1,"destRescName=*Res++++forceFlag=", *Status);
            msiSetACL("default","own","*Acct",*Dest1);
            msiDataObjChksum(*Dest1, "forceChksum=",*Chksum);
            if (*Check != *Chksum) {
                writeLine("*Lfile", "Bad checksum for file *Dest1");
            } else {
                writeLine("*Lfile", "Moved file *Src1 to *Dest1");
                writeLine("*Lfile", "Moved file *Src1 to *Dest1");
            }
        }
    }
}
INPUT *Res=$"stage", *DestZone =$"ornlZone", *Acct =$"Mauna-acct",
*Src=$"/Mauna/home/atmos/sensor", *Dest =*$"/DestZone/home/*Acct#Mauna/archive"
OUTPUT ruleExecOut

### 7.35 Integrity verification (Policy 18)
Integrity checks should be performed periodically to catch failure modes such as media failure, storage system failure, data overwrites, operator error, etc. Even if both the hardware and software perform flawlessly, it is still possible for an operator error to delete or overwrite a file. The replication rule is turned into a rule that is executed every year. A production capable version of the rule is shown that is restartable, monitors the execution rate, checks the input variables, maintains a log
file of all actions, repairs corrupted files, and replaces missing replicas. In the “delay” command, the execution frequency for repeating the rule needs to be set. An example for a test every 6 months would be:

delay(\\texttt{<PLUSET>1s</PLUSET>|<EF>6m</EF> } )

The rule is named “rda-replication-rule.r” and is listed in section 4.5.2. This rule uses the policy functions:

- checkCollInput
- checkRescInput
- createLogFile
- checkMetaExistsColl
- findZoneHostName
- isColl
- getNumSizeColl
- getRescColl
- selectRescUpdate
- createReplicas
- updateCollMeta

7.36 Technology management policies (Policy 49)
The izonereport command lists the properties of the data grid, including both the iCAT catalog and storage servers.

Updates about software versions and hardware versions can be tracked by periodically running the izonereport. The report includes information about micro-service plugins, policies, and storage systems.

7.37 Metadata catalog management (Policy 9)
The metadata catalog, iCAT, contains all of the state information for the data grid. To minimize risk, the metadata catalog should be replicated. Periodic backup dumps of the catalog should be saved outside of the data grid.

The data grid uses schema indirection to store descriptive and provenance metadata attributes. Once a standard schema is chosen, the schema can be installed as a HIVE ontology. A rule can then be run to compare the descriptive metadata for each file with the standard schema. An example rule is called validate-ontologies.r and is listed in section 5.12.1.

7.38 Transformative migration (Policy 15)
The migration of data formats to new technology is supported through invocation of external transformation systems, such as NCSA Polyglot and Brown Dog. Access to these systems is invoked through a micro-service that issues http post and get commands. Examples for invoking external services are listed in sections 5.13.1 (acPostProcForModifyAVUMetadata.r), 6.27 (hipaa-issue-url.r), and 7.9 (dmp-get-object-url.r).
8 Verifying Policy Sets:

To verify a theory of policy-based data management, a generic characterization of data management systems is needed. To base the discussion on well-known concepts, consider the characterization of file systems shown in Figure 2. The file system comprises an environment that is defined by the state information maintained about each file. Interactions with the file system consist of events that specify an operation. Each operation manipulates a file and changes the associated state information. Operations may require access to state information such as file location, or file size, or file owner. If the state information is consistently updated on each operation applied to files within the file system, the environment can have properties such as completeness, consistency, correctness, and closure. These properties describe four essential elements of data management:

1) What are the basic building blocks for composing procedures?
2) What are the constraints for procedure authoring and deployment?
3) How are procedures implemented?
4) How is the output of procedures handled?

Completeness means that all operations for each managed file type are supported. Consistency means that there are no conflicting procedures. Correctness means that a given operation performs without error. Closure means that operations on files will generate files that are members of the system. We can evaluate the properties of completeness, consistency, correctness, and closure by analyzing changes to the state information.

Typical file system state information is listed in Table 2. The operations performed upon the file system may consist of create, open, close, read, write, update, seek, stat, chown, link, and unlink. An operation may be applied to a file or to a group of files.

Interactions with the files are done through interactive execution of clients, which invoke the desired operation through a system call. This approach makes it possible to implement a standard data

![Figure 2. File System Characterization](image)

<table>
<thead>
<tr>
<th>Table 2. File System State Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name</td>
</tr>
<tr>
<td>File Location on disk</td>
</tr>
<tr>
<td>Creation time</td>
</tr>
<tr>
<td>Modification time</td>
</tr>
<tr>
<td>File size</td>
</tr>
<tr>
<td>Access control</td>
</tr>
<tr>
<td>Locks</td>
</tr>
<tr>
<td>Soft Link</td>
</tr>
<tr>
<td>Directory</td>
</tr>
</tbody>
</table>
management approach on different types of hardware systems, which in turn enables the migration of files across storage systems.

We can generalize this model of data management by introducing policies that control the operations performed within the system. In Figure 3, we introduce three significant changes:

- Operations are replaced by policies.
- Files are replaced by objects.
- Updates on objects and on state information are implemented as procedures.

A given event may invoke multiple policies. Each policy controls the execution of a procedure that chains together multiple operations expressed as micro-services. The objects manipulated by the policies can include resources, users, digital objects, micro-services, rules, metadata, and the properties of the environment itself.

For example, consider the addition of a file to the system. Even though the explicit event is a simple file addition, the response of the system may require the execution of multiple policies, with each policy potentially executing procedures that manipulate multiple types of objects. Policies that are executed may include:

1. Authentication of the person adding the file
2. Authorization for the addition of a file
3. Evaluation of a storage quota for the storage resource
4. Creation of a logical name for the file
5. Logical arrangement of the file as a member of a collection
6. Physical aggregation of the file into a container
7. Selection of a storage resource for the physical copy of the file
8. Creation of a physical file name on the storage resource
9. Inheritance of access controls from the collection access controls
10. Creation of a checksum
11. Replication of the file to a second storage location
12. Assignment of a retention period for the file
13. Assignment of a data type to the file based on the file extension
14. Storage of system level metadata (owner name, access controls, checksum, file size, replica location, retention period, file type)
15. Extraction and storage of descriptive metadata
16. Creation of an archival information package (aggregating metadata with the file)
17. Storage of the file

The response of the system is controlled by the policies that are enforced within the environment. A notable challenge is that policy-based data management systems have the ability to change the controlling policies, and therefore change the response of the system to external events. A process for validating the properties of the environment is needed to verify that either the new policies are compatible with prior policies and that the properties of the environment have not changed, or that the impact of the new policies can be defined as approved changes to state information.

We can characterize interactions with the data management system in terms of the allowed events. Events may be initiated interactively by external users, or by time-based procedures, or by changes of state information. In policy-based data management, events are detected at policy-enforcement points, which control the selection of policies that should be applied. The policies in turn control the execution of procedures that read/create/update state information and modify the objects in the system. Policies invoked at policy-enforcement points control how the environment responds to events. A mapping between events, the policy-enforcement points, the policies, the procedures, and associated changes to state information is necessary to describe the environment. If all changes to state information can be identified for all events, then the properties of the environment can be verified.

We can build a characterization of a data management system in terms of the following concepts:
1. Events invoked by users of the system
   a. Create, modify, delete, access
2. Entities that are managed by the system
   a. Users, digital objects, resources (storage, compute), metadata, rules, micro-services, environment framework
3. Policies that control assertions about the environment
   a. Properties associated with each type of entity (provenance metadata, access control, audit trail, aggregation, retention period)
   b. Properties controlling environment operations (number of processing threads, number of I/O streams, choice of physical path name)

We can verify a theory of policy-based data management by analyzing the consistency, completeness, correctness, and closure of the state information after application of every supported event. To do this we will need to define the set of policies that are invoked by each event. For each policy we will need to define the procedures that are invoked, and the set of state information variables that are modified by each procedure. Note that procedures are composed by chaining together micro-services. We can then identify the sets of state information
generated or modified by each micro-service. A verification policy can be defined that validates that the revised state information is consistent with the desired collection properties.

This approach can be applied for each data management domain (data sharing, digital library, preservation, processing pipeline) by analyzing the controlling policies and procedures. The results are domain dependent. An analysis needs to be done for each domain and for each change to the set of policies. However the approach is generic, and the underlying infrastructure that is used to implement the policy-based data management is generic.

In a distributed environment that encompasses multiple storage locations, multiple network paths, and multiple administrative domains, correctness cannot be guaranteed. A storage system may have a media failure and corrupt the data bits. A network may become unavailable and a transfer may not complete. A remote administrator may choose to perform maintenance and take an entire system offline. This implies that the environment needs to be able to detect inconsistencies, and use periodic policies to correct the problems. A simple example is the management of integrity. A standard approach is to generate a checksum for each file, and replicate the file across multiple storage systems. A policy can be executed periodically that verifies the integrity of each file by comparing the current checksum with the stored value. When a corrupted file is found, the system can delete the corrupted file, create a new replica from an uncorrupted copy, update the system metadata, and log the event. A goal in a policy-based data management system is to implement policies that verify the desired properties of the environment, and that implement recovery procedures as needed to ensure compliance. An extended goal is to implement policies that ensure that desired properties are maintained as the environment evolves.

8.1 Analysis of the integrated Rule Oriented Data System
The generality of the approach can be illustrated using the iRODS integrated Rule Oriented Data System [5, 10]. The iRODS software implements virtualization mechanisms that enable the federation of existing data management systems, and the enforcement of desired environment properties across the federated systems.

The iRODS data grid manages multiple types of entities independently of the choice of authentication environment, storage system, database, and administrative domain:

- Users (logical user name space)
- Digital objects (files, workflow structured objects, soft links)
- Resources (storage systems, repositories, compute systems)
- Metadata (system state information, provenance information, descriptive information)
- Rules (computer actionable policies that control the execution of procedures)
- Micro-services (computer executable functions that can be chained into procedures)
• Environment framework (the data grid itself).

Standard properties can be generated for each type of entity:
• Logical name (persistent identifier defined by the data grid)
• Access controls
• Aggregation (formation of groups)
• Descriptive metadata
• Audit trail of events and actions

The standard properties are reified as system state information that are stored in a relational database (the iCAT catalog). The impact of each event that accesses the system can be tracked through the corresponding changes to the state information. In iRODS, many of the state information attributes are updated by the iRODS server middleware to guarantee consistency. However the data grid administrator can customize changes to the system by modifying the policies that are stored in the rule base. Since these policies reflect decisions by the data grid administrator, a procedure is needed that verifies the consistency of the data grid.

We can generate a comprehensive assessment of the consistent update of state information by analyzing the mapping of:
• Events (client actions) to multiple policy-enforcement points
• Policies invoked at policy-enforcement points
• Procedures controlled by each policy
• Chain of micro-services invoked by a procedure
• Updates to state information generation by each micro-service
• Verification policy that monitors the state of the system

8.2 Policy-enforcement points
In Appendix A, we list the policy-enforcement points in iRODS. They can be loosely grouped into control points for manipulating files, users, resources, system state information, and environment parameters. While the iRODS data grid provides 71 policy-enforcement points, the standard data grid uses policies at only 11 points which are listed in section 2.

In practice, sites add rules to enforce specific properties within the data grid. For example, in the SILS LifeTime Library [11] five additional/modified rules are used, listed in section 3.

To verify that the LifeTime Library rule set enforces the required properties, we will need to examine which events invoke the policies, and then analyze changes to the state information for consistency.

8.3 Client invocation of policy-enforcement points
In Appendix B, we list events generated by the execution of the unix shell commands provided with the iRODS data grid (icommands). The unix shell commands are the most comprehensive interface for iRODS in terms of the policy-enforcement points
that can be triggered. Each command invocation may cause policies at multiple policy-enforcement points to be executed. For the case of loading a file into the data grid, the following ten policy-enforcement points are triggered:

1. acChkHostAccessControl
2. acSetPublicUserPolicy
3. acAclPolicy
4. acSetRescSchemeForCreate
5. acRescQuotaPolicy
6. acSetVaultPathPolicy
7. acPreProcForModifyDataObjMeta
8. acPostProcForModifyDataObjMeta
9. acPostProcForCreate
10. acPostProcForPut

We immediately can see that four of the policies added for the SILS LifeTime Library will need to be verified for their impact on policy-enforcement points 3, 4, 5, and 10 in the above list. The additional policy for the LifeTime Library controls the preferred storage location for replications. An assertion about the properties of the LifeTime Library requires verifying that the new policies have not changed the data grid properties. We do this by checking whether changes to the state information for each of these rules maintains the desired completeness, correctness, closure, and consistency.

A total of 80 different client interactions are listed in Appendix B, along with the policy enforcement points that are triggered.

For other events, a different set of policy enforcement points may be triggered. However, all clients (web browsers, load libraries, I/O libraries) will trigger the same policy enforcement points for the same events.

8.4 Procedures executed at each policy enforcement point

The procedures executed within the iRODS data grid are composed by chaining together micro-services. Appendix C lists the available micro-services, organized alphabetically. Most of the micro-services do not affect the system state information, and instead are used to manage the workflow, or interact with external systems, or support string manipulation, or support arithmetic operations, or support administrative functions. There are currently 348 micro-services available for use in rules.

For each micro-service the set of system attributes that are read, modified, or written is identified. A list of queriable persistent state information attributes are listed in Appendix D. If a persistent state information attribute is not included in Appendix C, then it is not read or modified by a micro-service. There are a total of 67 different sets of state information that may be modified. The sets are listed in tables C:2, C:3, and C:4.
Of the list of 348 micro-services, only 103 modify state information. Out of a total of 338 system state attributes, 151 attributes are modified by the micro-services.

The mapping challenge is therefore:

- 80 separate client events represented by icommand actions
- 71 policy enforcement points
- 103 micro-services that manipulate state information
- 151 persistent state attributes

The number of combinations that should checked is

\[
\text{Number of client events} \times \text{Number of policy enforcement points accessed by the event} \times \text{Number of micro-services invoked at a policy enforcement point} \times \text{Number of persistent state attributes modified by a micro-service.}
\]

In the following analysis, we ignore the policy-enforcement points that have not been modified, and the micro-services that are not invoked at a policy-enforcement point.

We examine the impact of each policy for the SILS LifeTime Library:

- acAclPolicy enforcement point is used by 37 of the client actions.
  - This policy calls the msiAclPolicy("STRICT") micro-service.
  - The msiAclPolicy sets “STRICT” access in a structure in memory. The persistent state information is not changed directly.
  - To check enforcement of this policy, a listing of files in a non-public user account can be tried to verify that the files cannot be seen.

- acSetRescSchemeForCreate enforcement point is used by 7 of the client actions, basically each time a file is created.
  - This policy calls the msiSetDefaultResc("lifelibResc1","null") micro-service.
  - The msiSetDefaultResc defines the storage system to use for creating a file in a structure in memory. The persistent state information is not changed directly.
  - The impact of the policy can be monitored by running a rule that verifies that each file has a copy residing on “lifelibResc1”:

```groovy
rule verifyFiles {
  # Verify each file has a copy on a specified storage resource
  *Path = "/$rodsZoneClient/home/$userNameClient/%";
  *Q = select DATA_NAME, COLL_NAME where COLL_NAME like '*Path';
  *Count = 0;
  foreach (*R in *Q) {
    *F = *R.DATA_NAME;
    *C = *R.COLL_NAME;
    *Q2 = select count(DATA_ID) where COLL_NAME = '*C' and DATA_NAME = '*F' and DATA_RESC_NAME = '*Resc';
    foreach (*R2 in *Q2) {
      if(*R2.DATA_ID == "0") {
        *Count = *Count + 1;
      }
    }
  }
}
```
acSetRescSchemeForRepl enforcement point is used by 1 client action for creating a replica.
  
- This policy also calls the msiSetDefaultResc("renci-unix1","null")
  micro-service.
- The msiSetDefaultResc defines the storage system to use for replicating a file in a structure in memory. The persistent state information is not changed directly.
- Enforcement of the policy can be monitored by running a rule that verifies that each file has a replica on "renci-unix1".

```plaintext
ruleverifyFiles {
  # Verify each file has a copy on a specified storage resource
  *Path = "/$rodsZoneClient/home/$userNameClient/%";
  *Q = select DATA_NAME, COLL_NAME where COLL_NAME like '*Path';
  *Count = 0;
  foreach (*R in *Q) {
    *F = *R.DATA_NAME;
    *C = *R.COLL_NAME;
    *Q2 = select count(DATA_ID) where COLL_NAME = '*C' and DATA_NAME = '*F' and DATA_RESC_NAME = '*Resc';
    foreach (*R2 in *Q2) {
      if(*R2.DATA_ID == "0") {
        *Count = *Count + 1;
      }
    }
  }
  writeLine("stdout", "A total of *Count files are not present on *Resc");
}
```

```
INPUT *Resc = "lifelibResc1"
OUTPUT ruleExecOut
```

acRescQuotaPolicy enforcement point is not called by an icommand.
  
- This policy calls the msiSetRescQuotaPolicy("on") micro-service.
- The msiSetRescQuotaPolicy turns on the storage quota in a structure in memory. The persistent state information is not changed directly.
- Enforcement of the policy can be checked by running a rule that checks the QUOTA_USAGE.

```plaintext
ruleQuota {
  # Count number of users that exceed the quota
  *Q = select QUOTA_USER_NAME, QUOTA_OVER;
  *Count = 0;
  foreach (*R in *Q) {
    *Over = double(*R.QUOTA_OVER);
    if(*Over > 0.) {
      *Count = *Count + 1;
      *User = *R.QUOTA_USER_NAME;
    }
  }
}
```

```
INPUT *Resc = "renci-unix1"
OUTPUT ruleExecOut
```
writeLine("stdout", "User *User exceeded quota");
}
writeLine("stdout", "Count persons exceed quota");
}

INPUT null
OUTPUT ruleExecOut

- acPostProcForPut enforcement point is used by 5 client actions.
  - The policy calls two micro-services
    - delay("<PLUSET>1s</PLUSET>")
      - This uses persistent state variable set #60 to modify state information:
        - RULE_EVENT
        - RULE_EXEC_ADDRESS
        - RULE_EXEC_ESTIMATED_EXE_TIME
        - RULE_EXEC_FREQUENCY
        - RULE_EXEC_ID
        - RULE_EXEC_NAME
        - RULE_EXEC_NOTIFICATION_ADDR
        - RULE_EXEC_PRIORITY
        - RULE_EXEC_REI_FILE_PATH
        - RULE_EXEC_TIME
        - RULE_EXEC_USER_NAME
        - RULE_ID
    - msiSysReplDataObj('renci-unix1','null')
      - This reads the persistent state variables in set #18 to collect state information:
        - COLL_CREATE_TIME
        - COLL_ID
        - COLL_MODIFY_TIME
        - COLL_NAME
        - COLL_OWNER_NAME
        - COLL_OWNER_ZONE
        - DATA_ACCESS_DATA_ID
        - DATA_ACCESS_TYPE
        - DATA_ACCESS_USER_ID
        - TOKEN_ID
        - TOKEN_NAME
        - TOKEN_NAMESPACE
        - USER_GROUP_ID
        - USER_ID
        - USER_NAME
        - USER_TYPE
        - USER_ZONE
  - This updates persistent state variables for the replica:
    - DATA_CHECKSUM
    - DATA_COLL_ID
    - DATA_COMMENTS
    - DATA_CREATE_TIME
    - DATA_EXPIRY
    - DATA_ID
    - DATA_MAP_ID
The creation of a replica can be verified by running a periodic rule that checks that a replica for each file exists, and that the integrity of the replica has not been compromised.
9 Summary:
The impact of modifications to the policies used in policy-based data management system can be based on analysis of changes to persistent state information. The process requires identifying the events (actions) executed by use of the system, and the responses made to the actions under policy-based control. The responses are mapped from the client events, through policy-enforcement points, to the policies that are enforced, to the micro-services that are executed, and finally to the persistent state information that is modified. Rules that analyze the consistency of the changed state information can then be periodically applied to verify system state. This approach requires an analysis rule for each policy that is changed. An example based on the SILS LifeTime Library policy set is presented.

10 Acknowledgements:
The development of the iRODS data grid and the research results in this paper were funded by the NSF OCI-1032732 grant, "SDCI Data Improvement: Improvement and Sustainability of iRODS Data Grid Software for Multi-Disciplinary Community Driven Application," (2010-2013), and the NSF Cooperative Agreement OCI-094084, "DataNet Federation Consortium", (2011-2015). We thank Shane Pusz, University of North Carolina at Chapel Hill for generating the micro-service usage information for the iRODS state information attributes.

11 References:
9. iRODS: https://www.irods.org

Appendix A: Policy-enforcement Points

Each policy-enforcement point is named. A policy can be added to the rule base (core.re file) using the name of a policy-enforcement point to invoke a controlling procedure. Thus to set access control to strict (meaning that no-one can see the names of anyone else’s files, we add the policy:

```
acAclPolicy {msiAclPolicy("STRICT"); }
```

The policy invokes the execution of the micro-service msiAclPolicy using the input parameter “STRICT”.

Three types of policy-enforcement points are used:

1. Provide control of the execution of a system function.
2. Provide pre-process control for defining input to the system function (acPreProc).
3. Provide post-process control for manipulating the output from the system function (acPostProc).

Table A.1 Policy Enforcement Points

<table>
<thead>
<tr>
<th>Policy Enforcement Point</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>acAclPolicy</td>
<td>This rule sets Access Control List policy.</td>
</tr>
<tr>
<td>acBulkPutPostProcPolicy</td>
<td>This rule sets the policy for executing the post processing put rule (acPostProcForPut) for bulk put.</td>
</tr>
<tr>
<td>acCheckPasswordStrength</td>
<td>This is a policy point for checking password strength, called when the admin or user is setting a password.</td>
</tr>
<tr>
<td>acChkHostAccessControl</td>
<td>This rule checks the access control by host and user based on the policy given in the HostAccessControl file.</td>
</tr>
<tr>
<td>acCreateDefaultCollections</td>
<td>This rule controls creation of standard collections for a new user.</td>
</tr>
<tr>
<td>acCreateUser</td>
<td>This rule enables pre-process and post-process for creation of a user.</td>
</tr>
<tr>
<td>acDataDeletePolicy</td>
<td>This rule sets the policy for deleting data objects. This is the PreProcessing rule for delete.</td>
</tr>
<tr>
<td>acDeleteUser</td>
<td>This rule enables pre-process and post-process for user deletion</td>
</tr>
<tr>
<td>acDeleteUserZoneCollections</td>
<td>This rule deletes standard user collections within a zone</td>
</tr>
<tr>
<td>acGetUserByDN</td>
<td>This rule can be configured to do some special handling of GSI DNs.</td>
</tr>
<tr>
<td>acPostProcForCollCreate</td>
<td>This rule sets the post-processing policy for creating a collection.</td>
</tr>
<tr>
<td>acPostProcForCopy</td>
<td>Rule for post processing the copy operation.</td>
</tr>
<tr>
<td>acPostProcForResource</td>
<td>Rule for post processing of data object create.</td>
</tr>
<tr>
<td>acPostProcForResourceGroup</td>
<td>This rule sets the post-processing policy for creating a new resource.</td>
</tr>
<tr>
<td>acPostProcForResourceGroup</td>
<td>This rule sets the post-processing policy for creating a new token.</td>
</tr>
<tr>
<td>acPostProcForCreateUser</td>
<td>This rule sets the post-processing policy for creating a new user.</td>
</tr>
<tr>
<td>acPostProcForDataObjRead</td>
<td>Rule for post processing the read buffer.</td>
</tr>
<tr>
<td>acPostProcForDataObjWrite</td>
<td>Rule for post processing the write buffer.</td>
</tr>
<tr>
<td>acPostProcForDelete</td>
<td>This rule sets the post-processing policy for deleting data objects.</td>
</tr>
<tr>
<td>acPostProcForDeleteResource</td>
<td>This rule sets the post-processing policy for deleting an old resource.</td>
</tr>
<tr>
<td>acPostProcForDeleteToken</td>
<td>This rule sets the post-processing policy for deleting an old token.</td>
</tr>
<tr>
<td>acPostProcForDeleteUser</td>
<td>This rule sets the post-processing policy for deleting an old user.</td>
</tr>
<tr>
<td>acPostProcForFilePathReg</td>
<td>Rule for post processing the registration or a file path.</td>
</tr>
<tr>
<td>acPostProcForGenQuery</td>
<td>This rule sets the post-processing policy for general query.</td>
</tr>
<tr>
<td>acPostProcForModifyAccessControl</td>
<td>This rule sets the post-processing policy for access control modification.</td>
</tr>
<tr>
<td>acPostProcForModifyAVUMetadata</td>
<td>This rule sets the post-processing policy for adding/deleting and copying the AVU metadata for data, collection, resources, and user.</td>
</tr>
<tr>
<td>acPostProcForModifyColMeta</td>
<td>This rule sets the post-processing policy for modifying system metadata of a collection.</td>
</tr>
<tr>
<td>acPostProcForModifyDataObjMeta</td>
<td>This rule sets the post-processing policy for modifying system metadata of a data object.</td>
</tr>
<tr>
<td>acPostProcForModifyResource</td>
<td>This rule sets the post-processing policy for modifying the properties of a resource.</td>
</tr>
<tr>
<td>acPostProcForModifyResourceGroup</td>
<td>This rule sets the post-processing policy for modifying membership of a resource.</td>
</tr>
<tr>
<td>Rule Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>acPostProcForModifyUser</code></td>
<td>This rule sets the post-processing policy for modifying the properties of a user.</td>
</tr>
<tr>
<td><code>acPostProcForModifyUserGroup</code></td>
<td>This rule sets the post-processing policy for modifying membership of a user group.</td>
</tr>
<tr>
<td><code>acPostProcForObjRename</code></td>
<td>This rule sets the post-processing policy for renaming (logically moving) data and collections.</td>
</tr>
<tr>
<td><code>acPostProcForOpen</code></td>
<td>Rule for post processing of data object open.</td>
</tr>
<tr>
<td><code>acPostProcForPhymv</code></td>
<td>Rule for post processing of data object move of a physical file path (e.g. -ireg command).</td>
</tr>
<tr>
<td><code>acPostProcForPut</code></td>
<td>Rule for post processing the put operation.</td>
</tr>
<tr>
<td><code>acPostProcForRmColl</code></td>
<td>This rule sets the post-processing policy for removing a collection.</td>
</tr>
<tr>
<td><code>acPostProcForTarFileReg</code></td>
<td>Rule for post processing the registration of the extracted tar file (from ibun -x).</td>
</tr>
<tr>
<td><code>acPreProcForCollCreate</code></td>
<td>This is the PreProcessing rule for creating a collection.</td>
</tr>
<tr>
<td><code>acPreProcForCreateResource</code></td>
<td>This rule sets the pre-processing policy for creating a new resource.</td>
</tr>
<tr>
<td><code>acPreProcForCreateToken</code></td>
<td>This rule sets the pre-processing policy for creating a new token.</td>
</tr>
<tr>
<td><code>acPreProcForCreateUser</code></td>
<td>This rule sets the pre-processing policy for creating a new user.</td>
</tr>
<tr>
<td><code>acPreProcForDataObjOpen</code></td>
<td>Preprocess rule for opening an existing data object which is used by the get, copy and replicate operations.</td>
</tr>
<tr>
<td><code>acPreProcForDeleteResource</code></td>
<td>This rule sets the pre-processing policy for deleting an old resource.</td>
</tr>
<tr>
<td><code>acPreProcForDeleteToken</code></td>
<td>This rule sets the pre-processing policy for deleting an old token.</td>
</tr>
<tr>
<td><code>acPreProcForDeleteUser</code></td>
<td>This rule sets the pre-processing policy for deleting an old user.</td>
</tr>
<tr>
<td><code>acPreProcForExecCmd</code></td>
<td>Rule for pre processing when remotely executing a command.</td>
</tr>
<tr>
<td><code>acPreProcForGenQuery</code></td>
<td>This rule sets the pre-processing policy for general query.</td>
</tr>
<tr>
<td><code>acPreProcForModifyAccessControl</code></td>
<td>This rule sets the pre-processing policy for access control modification.</td>
</tr>
<tr>
<td><code>acPreProcForModifyAVUmetadata</code></td>
<td>This rule sets the pre-processing policy for adding/deleting and copying the AVU metadata for data, collection, resources, and user.</td>
</tr>
<tr>
<td><code>acPreProcForModifyCollMeta</code></td>
<td>This rule sets the pre-processing policy for modifying system metadata of a collection.</td>
</tr>
<tr>
<td><code>acPreProcForModifyDataObjMeta</code></td>
<td>This rule sets the pre-processing policy for modifying system metadata of a data object.</td>
</tr>
<tr>
<td><code>acPreProcForModifyResource</code></td>
<td>This rule sets the pre-processing policy for modifying the properties of a resource.</td>
</tr>
<tr>
<td><code>acPreProcForModifyResourceGroup</code></td>
<td>This rule sets the pre-processing policy for modifying membership of a resource group.</td>
</tr>
<tr>
<td><code>acPreProcForModifyUser</code></td>
<td>This rule sets the pre-processing policy for modifying the properties of a user.</td>
</tr>
<tr>
<td><code>acPreProcForModifyUserGroup</code></td>
<td>This rule sets the pre-processing policy for modifying membership of a user group.</td>
</tr>
<tr>
<td><code>acPreProcForObjRename</code></td>
<td>This rule sets the pre-processing policy for renaming (logically moving) data and collections.</td>
</tr>
<tr>
<td><code>acPreProcForRmColl</code></td>
<td>This is the PreProcessing rule for removing a collection. Currently there is no function written specifically for this rule.</td>
</tr>
<tr>
<td><code>acRenameLocalZone</code></td>
<td>This rule renames the zone and all collections within the zone.</td>
</tr>
<tr>
<td><code>acRescQuotaPolicy</code></td>
<td>This rule sets the policy for a resource quota.</td>
</tr>
<tr>
<td><code>acSetChkFilePathPerm</code></td>
<td>This rule manages mounting of collections.</td>
</tr>
<tr>
<td><code>acSetMultReplPerResc</code></td>
<td>Preprocess rule for replicating an existing data object.</td>
</tr>
<tr>
<td><code>acSetNumThreads</code></td>
<td>Rule to set the number of threads for a data transfer.</td>
</tr>
<tr>
<td><code>acSetPublicUserPolicy</code></td>
<td>This rule sets the policy for the set of operations that are allowable for the user &quot;public&quot;.</td>
</tr>
<tr>
<td><code>acSetRescSchemeForCreate</code></td>
<td>This is the preprocessing rule for creating a data object.</td>
</tr>
<tr>
<td><code>acSetRescSchemeForRepl</code></td>
<td>This is the preprocessing rule for replicating a data object.</td>
</tr>
<tr>
<td><code>acSetReServerNumProc</code></td>
<td>This rule sets the policy for the number of processes to use when running jobs in the iRODSReServer.</td>
</tr>
<tr>
<td><code>acSetVaultPathPolicy</code></td>
<td>This rule sets the policy for creating the physical path in the iRODS resource vault.</td>
</tr>
<tr>
<td><code>acTicketPolicy</code></td>
<td>This is a policy point for ticket-based access control.</td>
</tr>
<tr>
<td><code>acTrashPolicy</code></td>
<td>This rule sets the policy for whether the trash can should be used.</td>
</tr>
</tbody>
</table>
Appendix B: Client Invocation of Policy Enforcement Points

Each policy enforcement point may be invoked by multiple client events. For events that manipulate files, up to 12 policy enforcement points are accessed for each interaction. In the following tables, the columns list the policy enforcement points. Client actions that invoke a policy enforcement point are listed in separate rows. Note that each table defines events that invoke different policy enforcement points.

<table>
<thead>
<tr>
<th>commands</th>
<th>acCheckHostAccessControl</th>
<th>acSetPublicUserPolicy</th>
<th>acSetBacSetCreatorPolicy</th>
<th>acSetBacSetOwnerPolicy</th>
<th>acSetBacSetReplicationPolicy</th>
<th>acSetFileObjectMetaPolicy</th>
<th>acSetFileObjectMetaCreatePolicy</th>
<th>acSetRescSchemeForCreate</th>
<th>acSetRescSchemeForReplic</th>
<th>acSetRescSchemeForRepl</th>
<th>acSetMultiReplPerResc</th>
<th>acSetVaultPathPolicy</th>
<th>acPreProcForDataObjOpen</th>
<th>acPostProcForDataObjOpen</th>
<th>acPreProcForModifyDataObjMeta</th>
<th>acPreProcForModifyDataObjMetaCreatePolicy</th>
<th>acPostProcForModifyDataObjMetaCreatePolicy</th>
<th>acTrashPolicy</th>
<th>acDataDeletePolicy</th>
<th>acPreProcForCollCreate</th>
<th>acPostProcForCollCreate</th>
<th>acPostProcForFilePathReg</th>
<th>acDataDeletePolicyCreate</th>
<th>acPostProcForFileRmCollCreate</th>
<th>acPostProcForFileRmCollDelete</th>
</tr>
</thead>
<tbody>
<tr>
<td>icp</td>
<td>Copy a file</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>icp -N 2</td>
<td>Copy a file using 2 I/O threads</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>iphybun</td>
<td>Physically bundle a collection</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>irepl</td>
<td>Replicate a file</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ibun -c D</td>
<td>Upload/download tar files</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>iput</td>
<td>Put a file into the data grid</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>iphyvm</td>
<td>Physically move a file</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>imv</td>
<td>Move a file</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>irm</td>
<td>Remove a file</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>irm - r collection</td>
<td>Recursively remove a collection</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ichksum</td>
<td>Checksum a file</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>iput -f</td>
<td>Overwrite an existing file</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>irsync</td>
<td>Synchronize two collections</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>irule - msiDataObjWrite</td>
<td>Write a file</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>irule - msiDataObjRead</td>
<td>Read a file</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>idbo</td>
<td>Execute a database resource</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>iget</td>
<td>Get a file from the data grid</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>igetwild.sh</td>
<td>Get multiple files</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>imkdir</td>
<td>Make a directory</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>ireg</td>
<td>Register a file</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>irmtrash</td>
<td>Empty trash</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>
### Table B.2 Events that manipulate users and resources

<table>
<thead>
<tr>
<th>Commands</th>
<th>acChkHostAccessControl</th>
<th>acSetPublicUserPolicy</th>
<th>acAclPolicy</th>
<th>acCreateUser</th>
<th>acPreProcForCreateUser</th>
<th>acCreateCollection</th>
<th>acDeleteUser</th>
<th>acPreProcForDeleteUser</th>
<th>acDeleteCollection</th>
</tr>
</thead>
<tbody>
<tr>
<td>iadmin mkuser</td>
<td>Make a user</td>
<td>x x x x x x x x x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin mkgroup</td>
<td>Make a user group</td>
<td>x x x x x x x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin moduser</td>
<td>Modify a user</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ipasswd</td>
<td>Create password</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin rmuser</td>
<td>Remove user</td>
<td>x x x x x x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin mkresc</td>
<td>Make a resource</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin rmresc</td>
<td>Remove a resource</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table B.3 Administrative Operations

<table>
<thead>
<tr>
<th>Commands</th>
<th>acChkHostAccessControl</th>
<th>acSetPublicUserPolicy</th>
<th>acAclPolicy</th>
<th>acCreateUser</th>
<th>acPreProcForCreateUser</th>
<th>acCreateCollection</th>
<th>acDeleteUser</th>
<th>acPreProcForDeleteUser</th>
<th>acDeleteCollection</th>
</tr>
</thead>
<tbody>
<tr>
<td>iadmin modresc</td>
<td>Modify a resource</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin atg</td>
<td>Add user to group</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin rfg</td>
<td>Remove use from group</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin atrg</td>
<td>Add resource to resource group</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin rfrg</td>
<td>Remove resource from resource group</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin at</td>
<td>Add token</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin rt</td>
<td>Remove token</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin pv</td>
<td>Initiate database vacuum</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>imeta</td>
<td>List metadata</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ichmod</td>
<td>Change access</td>
<td>x x</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>icoll -m</td>
<td>Mount a collection</td>
<td>x x x</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin modzone</td>
<td>Modify a zone</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irule -acPurgeFiles</td>
<td>Purge deleted files</td>
<td>x x</td>
<td>x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

218
<table>
<thead>
<tr>
<th>icommands</th>
<th>acChkHostAccessControl</th>
<th>acSetPublicUserPolicy</th>
<th>acCheckOwner</th>
<th>acGetUserByDN</th>
<th>acSetUserByDN</th>
<th>acGetSetInt</th>
<th>acSetReServerNumProc</th>
<th>acPreProcForGenQuery</th>
<th>acPostProcForGenQuery</th>
<th>acPostProcForDataObjWrite</th>
<th>acPostProcForDataObjRead</th>
</tr>
</thead>
<tbody>
<tr>
<td>irule - acConvertToInt</td>
<td>Execute a rule to convert to integer</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gsi authentication</td>
<td>Authenticate using GSI</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irule - acSetNumThreads</td>
<td>Set number of threads for data transfer</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irule - msiNoChkfilePathPerm.r</td>
<td>Set permissions for registration</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irule - acSetReServerNumProc</td>
<td>Set number of execution threads</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrePostProcForGenQueryFlag = 1</td>
<td>Execute general query</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReadWriteRuleState = ON_STATE</td>
<td>Modify a data object</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irule - rulemsiExecGenQuery</td>
<td>Execute a general query</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>init</td>
<td>Initialize access to the data grid</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin</td>
<td>Administration interface</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin mkdir</td>
<td>Make a directory</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>icd</td>
<td>Change directory</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irexecmd</td>
<td>Execute a remote command</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ifsck</td>
<td>Check consistency of data in vault</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ilocate</td>
<td>Search for a file</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ils</td>
<td>List files</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ilsresc</td>
<td>List resources</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>imiscsvrinfo</td>
<td>List server information</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ips</td>
<td>Display connections for running agents</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iqdel</td>
<td>Delete rule from queue</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iqmod</td>
<td>Modify rule in queue</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iqstat</td>
<td>List rules in queue</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iquest</td>
<td>Query metadata catalog</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iqquota</td>
<td>Show information on iRODS quotas</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>irule</td>
<td>Execute a rule</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iscan i:</td>
<td>Check registration of local files</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>isysmeta</td>
<td>List system metadata</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>itrim</td>
<td>Delete replicas</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iuserinfo</td>
<td>List user information</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ixmsg</td>
<td>Send a message</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ienv</td>
<td>List environment variables</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>help</td>
<td>List icommands</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin mkzone</td>
<td>Make a data grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin rmzone</td>
<td>Remove a data grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin asq</td>
<td>Set an alias</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iadmin rsq</td>
<td>Remove an alias</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ierror</td>
<td>List error message</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iexit</td>
<td>Exit from the data grid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ipwd</td>
<td>Change password</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C: Micro-services

The micro-services encapsulate basic operations that may be useful when implementing a policy. The types of operations include manipulation of:
1. Collections
2. Data objects
3. Output files and strings
4. Rule base
5. Workflow
6. Messaging system
7. Environment
8. Metadata
9. External services
10. Remote database access
11. Soft links
12. HDF
13. Property lists
14. URLs
15. Web services
16. XML

For each micro-service, an identifier is provided that defines the set of persistent state variables read or modified by execution of the micro-service. The persistent state variable sets are listed in Table C.2. Note that micro-services that do not modify state information are listed with persistent state set “0”.

Table C.1 List of micro-services available in iRODS version 4.0

<table>
<thead>
<tr>
<th>Micro-service</th>
<th>Persistent State Set</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-</code></td>
<td>Negation operator for arithmetic</td>
</tr>
<tr>
<td><code>!</code></td>
<td>Negation operator for boolean variables</td>
</tr>
<tr>
<td><code>!=</code></td>
<td>Negation operation for conditional test</td>
</tr>
<tr>
<td><code>.</code></td>
<td>Structure operator for extracting variables from structure</td>
</tr>
<tr>
<td><code>*</code></td>
<td>Workflow variable</td>
</tr>
<tr>
<td><code>/</code></td>
<td>Division operator for arithmetic</td>
</tr>
<tr>
<td><code>&amp;&amp;</code></td>
<td>And operator for query</td>
</tr>
<tr>
<td><code>%</code></td>
<td>Module operator for arithmetic</td>
</tr>
<tr>
<td><code>%%</code></td>
<td>Or operator for query</td>
</tr>
<tr>
<td><code>^</code></td>
<td>Exponentiation operator for arithmetic</td>
</tr>
<tr>
<td><code>^^</code></td>
<td>Calculate nth root for arithmetic</td>
</tr>
<tr>
<td><code>+</code></td>
<td>Addition operator for arithmetic</td>
</tr>
<tr>
<td><code>++</code></td>
<td>Addition operator for strings</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>Less than operator for conditional tests</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>less than or equal operator for conditional tests</td>
</tr>
<tr>
<td><code>=</code></td>
<td>Assignment operator for variables</td>
</tr>
<tr>
<td><code>==</code></td>
<td>Equal operator for conditional tests</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>Greater than operator for conditional tests</td>
</tr>
<tr>
<td><code>&gt;=</code></td>
<td>Greater than or equal operator for conditional tests</td>
</tr>
<tr>
<td>`</td>
<td></td>
</tr>
<tr>
<td><code>abs</code></td>
<td>Absolute value operator for arithmetic</td>
</tr>
<tr>
<td><code>applyAllRules</code></td>
<td>Apply all rules</td>
</tr>
<tr>
<td><code>average</code></td>
<td>Average operator for arithmetic</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>bool</td>
<td>Boolean type operator</td>
</tr>
<tr>
<td>break</td>
<td>Break loop execution operator for workflow</td>
</tr>
<tr>
<td>ceiling</td>
<td>Calculate closest larger integer for arithmetic</td>
</tr>
<tr>
<td>cons</td>
<td>List definition operator</td>
</tr>
<tr>
<td>cut</td>
<td>No retry operator on failure for workflow</td>
</tr>
<tr>
<td>datetime</td>
<td>Date-time converter for workflow</td>
</tr>
<tr>
<td>datetimef</td>
<td>Data-time formatted converter for workflow</td>
</tr>
<tr>
<td>delay</td>
<td>Delay execution of a rule</td>
</tr>
<tr>
<td>double</td>
<td>Double type operator</td>
</tr>
<tr>
<td>elem</td>
<td>List element operator</td>
</tr>
<tr>
<td>errorcode</td>
<td>Trap error code operator for workflow</td>
</tr>
<tr>
<td>errormsg</td>
<td>Trap error message operator for workflow</td>
</tr>
<tr>
<td>eval</td>
<td>Evaluate code</td>
</tr>
<tr>
<td>execCmdArg</td>
<td>Execute remote command with an argument</td>
</tr>
<tr>
<td>exp</td>
<td>Exponentiation operator for arithmetic</td>
</tr>
<tr>
<td>fail</td>
<td>Fail operator for workflow</td>
</tr>
<tr>
<td>floor</td>
<td>Calculate closest lower integer for arithmetic</td>
</tr>
<tr>
<td>for</td>
<td>For loop operator for workflow</td>
</tr>
<tr>
<td>foreach</td>
<td>For each loop operator for workflow list</td>
</tr>
<tr>
<td>hd</td>
<td>Calculate the head of a list</td>
</tr>
<tr>
<td>if</td>
<td>Conditional test for workflow</td>
</tr>
<tr>
<td>int</td>
<td>Integer type operator</td>
</tr>
<tr>
<td>let</td>
<td>Define function variables in an expression</td>
</tr>
<tr>
<td>like</td>
<td>Similarity operator for query</td>
</tr>
<tr>
<td>like regex</td>
<td>Similarity operator for query</td>
</tr>
<tr>
<td>list</td>
<td>List structure type</td>
</tr>
<tr>
<td>log</td>
<td>Logarithm operator for arithmetic</td>
</tr>
<tr>
<td>match</td>
<td>Matches a string against a regular expression</td>
</tr>
<tr>
<td>max</td>
<td>Maximum operator for arithmetic</td>
</tr>
<tr>
<td>min</td>
<td>Minimum operator for arithmetic</td>
</tr>
<tr>
<td>msiaclPolicy</td>
<td>Set access control policy</td>
</tr>
<tr>
<td>msiAddConditionToGenQuery</td>
<td>Add condition to a general query</td>
</tr>
<tr>
<td>msiAddKeyVal</td>
<td>Add key-value pair to an in-memory structure</td>
</tr>
<tr>
<td>msiAddKeyValToMspStr</td>
<td>Add key-value pair to an in-memory structure for concatenating command arguments</td>
</tr>
<tr>
<td>msiAddSelectFieldToGenQuery</td>
<td>Add select field to a general query</td>
</tr>
<tr>
<td>msiAddToNCA</td>
<td>Modify an array in a netCDF file</td>
</tr>
<tr>
<td>msiAddUserToGroup</td>
<td>Admin - add a user to a group</td>
</tr>
<tr>
<td>msiAddAppRuleStr</td>
<td>Admin - add rules to an in-memory structure</td>
</tr>
<tr>
<td>msiAddAppendToTopOfCoreRE</td>
<td>Admin - append rules to the top of the rule base (core.re file)</td>
</tr>
<tr>
<td>msiAddChangeCoreRE</td>
<td>Admin - change the rule base (core.re file)</td>
</tr>
<tr>
<td>msiAddClearAppRuleStruct</td>
<td>Admin - clear rules from the in-memory structure</td>
</tr>
<tr>
<td>msiAddInsertDVMapsFromStructIntoDB</td>
<td>Admin - Insert persistent state name maps from memory structure into database</td>
</tr>
<tr>
<td>msiAddInsertFNMapsFromStructIntoDB</td>
<td>Admin - Insert function names maps from memory structure into database</td>
</tr>
<tr>
<td>msiAddInsertMsrvcsFromStructIntoDB</td>
<td>Admin - insert micro-service names maps from in-memory structure into database</td>
</tr>
<tr>
<td>msiAddInsertRulesFromStructIntoDB</td>
<td>Admin - insert rules from memory structure into database</td>
</tr>
<tr>
<td>msiAddReadDVMapsFromFileIntoStruct</td>
<td>Admin - load persistent state name maps from file into memory structure</td>
</tr>
<tr>
<td>msiAddReadFNMapsFromFileIntoStruct</td>
<td>Admin - Load function name maps from file into memory structure</td>
</tr>
<tr>
<td>msiAddReadMsrvcsFromFileIntoStruct</td>
<td>Admin - Read micro-service name maps from file into memory structure</td>
</tr>
<tr>
<td>msiAddReadRulesFromFileIntoStruct</td>
<td>Admin - Read rules from file into memory structure</td>
</tr>
<tr>
<td>msiAddRetrieveRulesFromDBIntoStruct</td>
<td>Admin - Load rules from database into a memory structure</td>
</tr>
<tr>
<td>msiAddShowCoreRE</td>
<td>Admin - list rules from rule base (core.re file)</td>
</tr>
<tr>
<td>msiAddShowDVM</td>
<td>Admin - list persistent state names</td>
</tr>
<tr>
<td>msiAddShowFN</td>
<td>Admin - list function names (micro-services)</td>
</tr>
<tr>
<td>msiAddWriteDVMapsFromFileIntoFile</td>
<td>Admin - write persistent state name maps from memory into a file</td>
</tr>
<tr>
<td>msiAddWriteFNMapsFromFileIntoFile</td>
<td>Admin - write function name maps from memory into a file</td>
</tr>
<tr>
<td>msiAddWriteMsrvcsFromFileIntoFile</td>
<td>Admin - write micro-service names maps from memory into a file</td>
</tr>
<tr>
<td>msiAddWriteRulesFromFileIntoFile</td>
<td>Admin - write rules from memory into a file</td>
</tr>
<tr>
<td>msiApplyDCMetadataTemplate</td>
<td>Apply the Dublin Core template to set attribute-value-unit triplets on a digital object</td>
</tr>
<tr>
<td>Function Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>msiAssociateKeyValuePairsToObject</td>
<td>Add attribute-value-units to a digital object, specified as key-value pairs</td>
</tr>
<tr>
<td>msiAutoReplicateService</td>
<td>Verify integrity and repair corrupted digital objects</td>
</tr>
<tr>
<td>msiBytesBufToStr</td>
<td>Format a buffer into a string</td>
</tr>
<tr>
<td>msiChecksumAccess</td>
<td>Check access control</td>
</tr>
<tr>
<td>msiChecksumHostAccessControl</td>
<td>Check host access control</td>
</tr>
<tr>
<td>msiCheckOwner</td>
<td>Check owner of a digital object</td>
</tr>
<tr>
<td>msiCheckPermission</td>
<td>Check access permissions</td>
</tr>
<tr>
<td>msiCloseGenQuery</td>
<td>Close the memory structure for a general query</td>
</tr>
<tr>
<td>msiCollCreate</td>
<td>Create a collection</td>
</tr>
<tr>
<td>msiCollectionSpider</td>
<td>Apply workflow to digital objects in a collection</td>
</tr>
<tr>
<td>msiCollRepl</td>
<td>Replicate a collection</td>
</tr>
<tr>
<td>msiCollRsync</td>
<td>Recursively synchronize a source collection with a target collection</td>
</tr>
<tr>
<td>msiCommit</td>
<td>Commit a change to the metadata catalog</td>
</tr>
<tr>
<td>msiConvertCurrency</td>
<td>Get conversion rates for currencies from a web service</td>
</tr>
<tr>
<td>msiCopyAVUMetadata</td>
<td>Copy attribute-value-units between digital objects</td>
</tr>
<tr>
<td>msiCreateCollByAdmin</td>
<td>Admin - create a collection</td>
</tr>
<tr>
<td>msiCreateUser</td>
<td>Admin - create a user</td>
</tr>
<tr>
<td>msiCreateUserAccountsFromDataObject</td>
<td>Create user accounts specified in a list in a digital object</td>
</tr>
<tr>
<td>msiCreateXmsgInp</td>
<td>Create an Xmsg packet from input parameters (messaging system)</td>
</tr>
<tr>
<td>msiCutBufferInHalf</td>
<td>Decrease size of an in-memory buffer</td>
</tr>
<tr>
<td>msiDataObjAutoMove</td>
<td>Move a file into a destination collection</td>
</tr>
<tr>
<td>msiDataObjChksum</td>
<td>Checksum a digital object</td>
</tr>
<tr>
<td>msiDataObjClose</td>
<td>Close a digital object</td>
</tr>
<tr>
<td>msiDataObjCopy</td>
<td>Copy a digital object</td>
</tr>
<tr>
<td>msiDataObjCreate</td>
<td>Create a digital object</td>
</tr>
<tr>
<td>msiDataObjGet</td>
<td>Get a digital object</td>
</tr>
<tr>
<td>msiDataObjLseek</td>
<td>Seek to a location in a digital object</td>
</tr>
<tr>
<td>msiDataObjOpen</td>
<td>Open a digital object</td>
</tr>
<tr>
<td>msiDataObjPhymv</td>
<td>Physically move a digital object</td>
</tr>
<tr>
<td>msiDataObjPut</td>
<td>Put a digital object into the data grid</td>
</tr>
<tr>
<td>msiDataObjRead</td>
<td>Read a digital object</td>
</tr>
<tr>
<td>msiDataObjRename</td>
<td>Rename a digital object</td>
</tr>
<tr>
<td>msiDataObjRepl</td>
<td>Replicate a digital object</td>
</tr>
<tr>
<td>msiDataObjRsync</td>
<td>Synchronize a digital object with an iRODS collection</td>
</tr>
<tr>
<td>msiDataObjTrim</td>
<td>Delete selected replicas of a digital object</td>
</tr>
<tr>
<td>msiDataObjUnlink</td>
<td>Delete a digital object</td>
</tr>
<tr>
<td>msiDataObjWrite</td>
<td>Write a digital object</td>
</tr>
<tr>
<td>msiDbExec</td>
<td>Execute a database resource object</td>
</tr>
<tr>
<td>msiDbrCommit</td>
<td>Execute a database resource commit</td>
</tr>
<tr>
<td>msiDbrollback</td>
<td>Rollback a database resource object</td>
</tr>
<tr>
<td>msiDeleteCollByAdmin</td>
<td>Admin - delete a collection</td>
</tr>
<tr>
<td>msiDeleteDisallowed</td>
<td>Turn off deletion for a digital object</td>
</tr>
<tr>
<td>msiDeleteUnusedAVUs</td>
<td>Delete unused attribute-value-unit triplets</td>
</tr>
<tr>
<td>msiDeleteUser</td>
<td>Delete a user</td>
</tr>
<tr>
<td>msiDeleteUsersFromDataObject</td>
<td>Delete users specified in a list in a digital object</td>
</tr>
<tr>
<td>msiDigestMonStat</td>
<td>Generate and store load factors for monitoring resources</td>
</tr>
<tr>
<td>msiDoSomething</td>
<td>Template for constructing a new micro-service</td>
</tr>
<tr>
<td>msiExecCmd</td>
<td>Execute a remote command</td>
</tr>
<tr>
<td>msiExecStrCondQuery</td>
<td>Execute general query</td>
</tr>
<tr>
<td>msiExecStrCondQuery</td>
<td>Convert a string to a query and execute</td>
</tr>
<tr>
<td>msiExit</td>
<td>Add a user explanation to the error stack</td>
</tr>
<tr>
<td>msiExportRecursiveCollMeta</td>
<td>Recursively export collection metadata into a buffer using pipe-delimited format</td>
</tr>
<tr>
<td>msiExtractTemplateMDFromBuf</td>
<td>Use a template to apply pattern matching to a buffer and extract key-value pairs</td>
</tr>
<tr>
<td>msiFlagDataObjectWithAVU</td>
<td>Add an attribute-value-unit to a digital object</td>
</tr>
<tr>
<td>msiFlagInfectedObjs</td>
<td>Parse the output from clamscan and flag infected objects</td>
</tr>
<tr>
<td>msiFloatToQString</td>
<td>Convert a binary variable to a string</td>
</tr>
<tr>
<td>msiFlushMonStat</td>
<td>Delete old usage monitoring statistics</td>
</tr>
<tr>
<td>msiFreeBuffer</td>
<td>Free space allocated to an in-memory buffer</td>
</tr>
<tr>
<td>msiFreeNcStruct</td>
<td>Free an in-memory structure used to process netCDF files</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>msiFtpGet</td>
<td>Get a file from an FTP site</td>
</tr>
<tr>
<td>msiGetAuditTrailInfoByActionID</td>
<td>Get audit trail information based on ActionID</td>
</tr>
<tr>
<td>msiGetAuditTrailInfoByKeywords</td>
<td>Get audit trail information based on use of keywords</td>
</tr>
<tr>
<td>msiGetAuditTrailInfoByObjectID</td>
<td>Get audit trail information based on ObjectIDs</td>
</tr>
<tr>
<td>msiGetAuditTrailInfoByTimeStamp</td>
<td>Get audit trail information based on time stamps</td>
</tr>
<tr>
<td>msiGetAuditTrailInfoByUserID</td>
<td>Get audit trail information based on userID</td>
</tr>
<tr>
<td>msiGetCollectionACL</td>
<td>Get access controls for a collection</td>
</tr>
<tr>
<td>msiGetCollectionContentsReport</td>
<td>Generate a report of collection contents</td>
</tr>
<tr>
<td>msiGetCollectionPSmeta</td>
<td>Get attribute-value-units from a collection in pipe-delimited format</td>
</tr>
<tr>
<td>msiGetCollectionSize</td>
<td>Get the size of a collection</td>
</tr>
<tr>
<td>msiGetContinIdxFromGenQueryOut</td>
<td>Get continuation index for whether additional rows are available for a query result</td>
</tr>
<tr>
<td>msiGetDataObjACL</td>
<td>Get access control list for a digital object</td>
</tr>
<tr>
<td>msiGetDataObjAIP</td>
<td>Create XML file containing system and descriptive metadata</td>
</tr>
<tr>
<td>msiGetDataObjAVUs</td>
<td>Get attribute-value-units from a digital object</td>
</tr>
<tr>
<td>msiGetDataObjPSmeta</td>
<td>Get attribute-value-units from a digital object in pipe-delimited format</td>
</tr>
<tr>
<td>msiGetDiffTime</td>
<td>Get the difference between two system times</td>
</tr>
<tr>
<td>msiGetDVMapsFromDBIntoStruct</td>
<td>Load persistent state name maps from database into memory structure</td>
</tr>
<tr>
<td>msiGetFNMapsFromDBIntoStruct</td>
<td>Load function name maps from database into memory structure</td>
</tr>
<tr>
<td>msiGetCatTime</td>
<td>Get the system time from the metadata catalog</td>
</tr>
<tr>
<td>msiGetMoreRows</td>
<td>Get more query results</td>
</tr>
<tr>
<td>msiGetMSrvcsFromDBIntoStruct</td>
<td>Load micro-service names from database into memory structure</td>
</tr>
<tr>
<td>msiGetObjectPath</td>
<td>Convert from in-memory structure to string for printing</td>
</tr>
<tr>
<td>msiGetObjType</td>
<td>Get the type of digital object (file, collection, user, resource)</td>
</tr>
<tr>
<td>msiGetQuote</td>
<td>Get stock quotation by accessing external web service</td>
</tr>
<tr>
<td>msiGetRescAddr</td>
<td>Get the IP address of a storage resource</td>
</tr>
<tr>
<td>msiGetRulesFromDBIntoStruct</td>
<td>Load rules from database into a memory structure</td>
</tr>
<tr>
<td>msiGetSessionVarValue</td>
<td>Get value of a session variable from in-memory structure</td>
</tr>
<tr>
<td>msiGetStdoutInExecCmdOut</td>
<td>Retrieve standard error from remote command execution</td>
</tr>
<tr>
<td>msiGetStdoutOutExecCmdOut</td>
<td>Retrieve standard output from remote command execution</td>
</tr>
<tr>
<td>msiGetSystemTime</td>
<td>Get the system time from the iRODS server</td>
</tr>
<tr>
<td>msiGetTaggedValueFromString</td>
<td>Use pattern-based extraction to retrieve a value for a tag from a string</td>
</tr>
<tr>
<td>msiGetUserACL</td>
<td>Get access control list for a user</td>
</tr>
<tr>
<td>msiGetUserInfo</td>
<td>Get information about a user</td>
</tr>
<tr>
<td>msiGetValByKey</td>
<td>Extract a value from in-memory structure that holds result of a query</td>
</tr>
<tr>
<td>msiGoodFailure</td>
<td>Force failure in a workflow without initiating recovery procedures</td>
</tr>
<tr>
<td>msiGuessDataType</td>
<td>Guess the data type based on the file extension</td>
</tr>
<tr>
<td>msiH5Dataset_read</td>
<td>Read an HDF5 file</td>
</tr>
<tr>
<td>msiH5Dataset_read_attribute</td>
<td>Get attributes from an HDF5 file</td>
</tr>
<tr>
<td>msiH5File_close</td>
<td>Close an HDF5 file</td>
</tr>
<tr>
<td>msiH5File_open</td>
<td>Open an HDF5 file</td>
</tr>
<tr>
<td>msiH5Group_read_attribute</td>
<td>Get group attributes from an HDF5 file</td>
</tr>
<tr>
<td>msiHumanToSystemTime</td>
<td>Convert human time format to system time format</td>
</tr>
<tr>
<td>msiImageConvert</td>
<td>Convert image format</td>
</tr>
<tr>
<td>msiImageGetProperties</td>
<td>Get image properties from an image (Colors, ColorSpace, Depth, Format, Gamma, ...)</td>
</tr>
<tr>
<td>msiIp2location</td>
<td>Convert an IP address to a location using an external web service</td>
</tr>
<tr>
<td>msiIsColl</td>
<td>Verify digital object is a collection</td>
</tr>
<tr>
<td>msiIsData</td>
<td>Check if digital object is a file</td>
</tr>
<tr>
<td>msiistEnabledMS</td>
<td>List enabled micro-services</td>
</tr>
<tr>
<td>msiLoadACLsFromDataObj</td>
<td>Load access controls from a list in a digital object</td>
</tr>
<tr>
<td>msiLoadMetadataFromDataObj</td>
<td>Load attribute-value-units from a list in a digital object</td>
</tr>
<tr>
<td>msiLoadMetadataFromXml</td>
<td>Load metadata for digital objects from an XML file</td>
</tr>
<tr>
<td>msiLoadUserModsFromDataObj</td>
<td>Load user information from a list in a digital object</td>
</tr>
<tr>
<td>msiMakeGenQuery</td>
<td>Make a general query</td>
</tr>
<tr>
<td>msiMakeQuery</td>
<td>Construct a query</td>
</tr>
<tr>
<td>msiMergeDataCopies</td>
<td>Merge multiple collections to create an authoritative version</td>
</tr>
<tr>
<td>msiNccGetVara</td>
<td>Get variables from a netCDF file</td>
</tr>
<tr>
<td>msiNccGetVara</td>
<td>Get a netCDF file</td>
</tr>
<tr>
<td>msiNccCreate</td>
<td>Create a netCDF file</td>
</tr>
<tr>
<td>msiNccGetArrayLen</td>
<td>Get array length from a netCDF file</td>
</tr>
<tr>
<td>msiNccGetAttNameInqOut</td>
<td>Get attribute names from a netCDF file</td>
</tr>
<tr>
<td>msiNccGetAttValInqOut</td>
<td>Get attribute values from a netCDF file</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>msiNcGetDataType</td>
<td>Get data type from a netCDF file</td>
</tr>
<tr>
<td>msiNcGetDimLenInInqOut</td>
<td>Get dimension length from a netCDF file</td>
</tr>
<tr>
<td>msiNcGetDimNameInInqOut</td>
<td>Get dimension name from a netCDF file</td>
</tr>
<tr>
<td>msiNcGetElementInArray</td>
<td>Get an element from an array in a netCDF file</td>
</tr>
<tr>
<td>msiNcGetFormatInInqOut</td>
<td>Get the format of a netCDF file</td>
</tr>
<tr>
<td>msiNcGetGrpInInqOut</td>
<td>Get group information from a netCDF file</td>
</tr>
<tr>
<td>msiNcGetNattsInInqOut</td>
<td>Get the number of attributes in a netCDF file</td>
</tr>
<tr>
<td>msiNcGetNdimsInInqOut</td>
<td>Get the number of dimensions in a netCDF file</td>
</tr>
<tr>
<td>msiNcGetNgrpsInInqOut</td>
<td>Get the number of groups in a netCDF file</td>
</tr>
<tr>
<td>msiNcGetNumDim</td>
<td>Get a dimension from a netCDF file</td>
</tr>
<tr>
<td>msiNcGetNvarsInInqOut</td>
<td>Get the number of variables in a netCDF file</td>
</tr>
<tr>
<td>msiNcGetVarIdInInqOut</td>
<td>Get a variable ID from a netCDF file</td>
</tr>
<tr>
<td>msiNcGetVarNameInInqOut</td>
<td>Get a variable name from a netCDF file</td>
</tr>
<tr>
<td>msiNcGetVarsByType</td>
<td>General variable sub-setting function for a netCDF file</td>
</tr>
<tr>
<td>msiNcGetVarTypeInInqOut</td>
<td>Get a variable type from a netCDF file</td>
</tr>
<tr>
<td>msiNcInq</td>
<td>Query a netCDF file</td>
</tr>
<tr>
<td>msiNcInqGrps</td>
<td>Get group paths for a given netCDF ID</td>
</tr>
<tr>
<td>msiNcInqId</td>
<td>Get netCDF ID</td>
</tr>
<tr>
<td>msiNcInqWithId</td>
<td>Query a netCDF file with a netCDF ID</td>
</tr>
<tr>
<td>msiNcIntDataTypeToStr</td>
<td>Convert netCDF data type to a string</td>
</tr>
<tr>
<td>msiNcOpen</td>
<td>Open a netCDF file</td>
</tr>
<tr>
<td>msiNcOpenGroup</td>
<td>Open a group within a netCDF file</td>
</tr>
<tr>
<td>msiNcRegGlobalAttr</td>
<td>Register a global attribute in a netCDF file</td>
</tr>
<tr>
<td>msiNcSubsetVar</td>
<td>Subset a variable in a netCDF file</td>
</tr>
<tr>
<td>msiNcVarStat</td>
<td>List variable information in a netCDF file</td>
</tr>
<tr>
<td>msiNoChkFilePathPerm</td>
<td>Set policy for checking the file path permission when registering a physical file path</td>
</tr>
<tr>
<td>msiNoTrashCan</td>
<td>Set policy for use of trash can</td>
</tr>
<tr>
<td>msiObjByName</td>
<td>Retrieve astronomy images by name using web services</td>
</tr>
<tr>
<td>msiobjGet_dbo</td>
<td>Get a database object from a registered database resource</td>
</tr>
<tr>
<td>msiobjGet_http</td>
<td>Get an http page from a registered web site</td>
</tr>
<tr>
<td>msiobjGet_irods</td>
<td>Get a file from a registered iRODS path name</td>
</tr>
<tr>
<td>msiobjGet_slink</td>
<td>Get a digital object referenced by a soft link to an iRODS data grid</td>
</tr>
<tr>
<td>msiobjGet_srb</td>
<td>Get a file from a registered Storage Resource Broker path name</td>
</tr>
<tr>
<td>msiobjGet_test</td>
<td>Test the micro-service object framework</td>
</tr>
<tr>
<td>msiobjGet_z3950</td>
<td>Get an object from a registered Z39.50 site</td>
</tr>
<tr>
<td>msiobjPut_dbo</td>
<td>Write a registered database object resource</td>
</tr>
<tr>
<td>msiobjPut_http</td>
<td>Write a registered http page</td>
</tr>
<tr>
<td>msiobjPut_irods</td>
<td>Write a registered iRODS digital object</td>
</tr>
<tr>
<td>msiobjPut_slink</td>
<td>Write a registered iRODS digital object in a remote iRODS data grid</td>
</tr>
<tr>
<td>msiobjPut_srb</td>
<td>Write a registered Storage Resource Broker digital object</td>
</tr>
<tr>
<td>msiobjPut_test</td>
<td>Test the micro-service object framework</td>
</tr>
<tr>
<td>msiobjPut_z3950</td>
<td>Write a registered 3 39.50 digital object</td>
</tr>
<tr>
<td>msiObjStat</td>
<td>Get status of digital object for workflow</td>
</tr>
<tr>
<td>msiOprDisallow</td>
<td>Disallow an operation</td>
</tr>
<tr>
<td>msiPhyBundleColl</td>
<td>Physically bundle a collection</td>
</tr>
<tr>
<td>msiPhyPathReg</td>
<td>Register a physical path</td>
</tr>
<tr>
<td>msiPrintGenQueryInp</td>
<td>Print a general query</td>
</tr>
<tr>
<td>msiPrintGenQueryOutToBuffer</td>
<td>Write contents of output results from a general query into a buffer</td>
</tr>
<tr>
<td>msiPrintKeyValPair</td>
<td>Print a key value pair returned from a query</td>
</tr>
<tr>
<td>msiPropertiesAdd</td>
<td>Add properties to a list</td>
</tr>
<tr>
<td>msiPropertiesClear</td>
<td>Clear properties from a list</td>
</tr>
<tr>
<td>msiPropertiesClone</td>
<td>Clone a properties list</td>
</tr>
<tr>
<td>msiPropertiesExists</td>
<td>Verify existence of properties in a list</td>
</tr>
<tr>
<td>msiPropertiesFromString</td>
<td>Create a properties list from a string</td>
</tr>
<tr>
<td>msiPropertiesGet</td>
<td>Get a property from a list</td>
</tr>
<tr>
<td>msiPropertiesNew</td>
<td>Create a new property list</td>
</tr>
<tr>
<td>msiPropertiesRemove</td>
<td>Remove properties from a list</td>
</tr>
<tr>
<td>msiPropertiesSet</td>
<td>Set the value of a property in a list</td>
</tr>
<tr>
<td>msiPropertiesToString</td>
<td>Convert a property list into a string buffer</td>
</tr>
<tr>
<td>msiQuota</td>
<td>Admin - calculate storage usage and check storage quotas</td>
</tr>
<tr>
<td>msiRcvXmsg</td>
<td>Receive an Xmsg packet (messaging system)</td>
</tr>
<tr>
<td>msiReadMDTemplateIntoTagStruct</td>
<td>Parse a buffer holding a tag template and store the tags in an in-memory</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>msiRecursiveCollCopy</td>
<td>Recursively copy a collection</td>
</tr>
<tr>
<td>msiRemoveKeyValuePairsFromObj</td>
<td>Remove attribute-value-unit from digital object, specified as key-value pair</td>
</tr>
<tr>
<td>msiRenameCollection</td>
<td>Rename a collection</td>
</tr>
<tr>
<td>msiRenameLocalZone</td>
<td>Admin - Rename the local zone (data grid)</td>
</tr>
<tr>
<td>msiRmColl</td>
<td>Remove a collection</td>
</tr>
<tr>
<td>msiRollback</td>
<td>Roll back a database transaction</td>
</tr>
<tr>
<td>msiSdSimgCutout_GetJpeg</td>
<td>Get an astronomy image cutout using a web service</td>
</tr>
<tr>
<td>msiSendMail</td>
<td>Send e-mail message</td>
</tr>
<tr>
<td>msiSendStdoutAsEmail</td>
<td>Send standard output as an e-mail message</td>
</tr>
<tr>
<td>msiSendXmsg</td>
<td>Send an Xmsg packet (messaging system)</td>
</tr>
<tr>
<td>msiServerBackup</td>
<td>Backup an iRODS server to a local vault</td>
</tr>
<tr>
<td>msiServerMonPerf</td>
<td>Monitor server performance</td>
</tr>
<tr>
<td>msiSetACL</td>
<td>Set an access control</td>
</tr>
<tr>
<td>msiSetBulkPutPostProcPolicy</td>
<td>Control use of the acPostProcForPut policy when using a bulk put operation</td>
</tr>
<tr>
<td>msiSetChkFilePathPerm</td>
<td>Disallow non-admin user from registering files</td>
</tr>
<tr>
<td>msiSetDataObjAvoidResc</td>
<td>Disallow use of a storage resource</td>
</tr>
<tr>
<td>msiSetDataObjPreferredResc</td>
<td>Set the preferred storage resource</td>
</tr>
<tr>
<td>msiSetDataType</td>
<td>Set the type of digital object (file, collection, user, resource)</td>
</tr>
<tr>
<td>msiSetDataTypeFromExt</td>
<td>Set a recognized data type for a digital object based on its extension</td>
</tr>
<tr>
<td>msiSetDefaultResc</td>
<td>Set the default storage resource</td>
</tr>
<tr>
<td>msiSetGraftPathScheme</td>
<td>Define the physical path name for storing files</td>
</tr>
<tr>
<td>msiSetMultiRepPerResc</td>
<td>Allow multiple replicas to exist on the same storage resource</td>
</tr>
<tr>
<td>msiSetNoDirectRescInp</td>
<td>Define a list of resources that cannot be used by a normal user</td>
</tr>
<tr>
<td>msiSetNumThreads</td>
<td>Set the number of threads used for parallel I/O</td>
</tr>
<tr>
<td>msiSetPublicUserOpr</td>
<td>Set a list of operations that can be performed by the user &quot;public&quot;</td>
</tr>
<tr>
<td>msiSetQuota</td>
<td>Set resource usage quota</td>
</tr>
<tr>
<td>msiSetRandomScheme</td>
<td>Set the physical path name based on a randomly generated path</td>
</tr>
<tr>
<td>msiSetRepComment</td>
<td>Set data object comment field</td>
</tr>
<tr>
<td>msiSetRescQuotaPolicy</td>
<td>Turn resource quotas on or off</td>
</tr>
<tr>
<td>msiSetRescSortScheme</td>
<td>Set the scheme used for selecting a storage resource</td>
</tr>
<tr>
<td>msiSetReServerNumProc</td>
<td>Set the number of execution threads for processing rules</td>
</tr>
<tr>
<td>msiSetResource</td>
<td>Set the resource to use within a workflow</td>
</tr>
<tr>
<td>msiSleep</td>
<td>Sleep for a specified interval</td>
</tr>
<tr>
<td>msiSortDataObj</td>
<td>Sort the order in which resources will be accessed to retrieve a replicated</td>
</tr>
<tr>
<td>msiSplitPath</td>
<td>Split a path into a collection and file name</td>
</tr>
<tr>
<td>msiSplitPathByKey</td>
<td>Split a path based on a key (separate a file name from an extension)</td>
</tr>
<tr>
<td>msiStageDataObj</td>
<td>Stage a digital object to a specified resource</td>
</tr>
<tr>
<td>msiStoreVersionWithTS</td>
<td>Create a time-stamped version of a digital object</td>
</tr>
<tr>
<td>msiStrArray2String</td>
<td>Convert an array of strings to a list of strings separated by &quot;%&quot;</td>
</tr>
<tr>
<td>msiStrCat</td>
<td>Concatenate a string to a target string</td>
</tr>
<tr>
<td>msiStrchop</td>
<td>Remove the last character of a string</td>
</tr>
<tr>
<td>msiString2KeyValPair</td>
<td>Convert a string to a key-value pair in memory structure</td>
</tr>
<tr>
<td>msiString2StrArray</td>
<td>Convert a list of strings separated by &quot;%&quot; to an in-memory array of strings</td>
</tr>
<tr>
<td>msiStripAVUs</td>
<td>Remove attribute-value-units from a digital object</td>
</tr>
<tr>
<td>msiStrlen</td>
<td>Get the length of a string</td>
</tr>
<tr>
<td>msiStrToBytesBuf</td>
<td>Load a string into an in-memory buffer</td>
</tr>
<tr>
<td>msiStructFileBundle</td>
<td>Create a bundle of files in a collection for export as a tar file</td>
</tr>
<tr>
<td>msiSysChksumDataObj</td>
<td>Checksum a digital object</td>
</tr>
<tr>
<td>msiSysMetaModify</td>
<td>Modify system metadata attributes</td>
</tr>
<tr>
<td>msiSysReplDataObj</td>
<td>Admin - replicate a digital object</td>
</tr>
<tr>
<td>msiTarFileCreate</td>
<td>Create a tar file</td>
</tr>
<tr>
<td>msiTarFileExtract</td>
<td>Extract files from a tar file</td>
</tr>
<tr>
<td>msiVacuum</td>
<td>Optimize indices in the metadata catalog</td>
</tr>
<tr>
<td>msiWriterRodsLog</td>
<td>Write a string into iRODS/server/log/rodsLog</td>
</tr>
<tr>
<td>msiXmlDocSchemaValidate</td>
<td>Validate an XML document schema for adding attributed-value-unit triplets</td>
</tr>
<tr>
<td>msiXmsgCreateStream</td>
<td>Create a message stream (messaging system)</td>
</tr>
<tr>
<td>msiXmsgServerConnect</td>
<td>Connect to a message stream (messaging system)</td>
</tr>
<tr>
<td>msiXmsgServerDisConnect</td>
<td>Disconnect from a message stream (messaging system)</td>
</tr>
</tbody>
</table>
The sets of persistent state information are listed in table C:2. Each persistent state information set identifies whether a persistent state:

- 1 – attribute is read
- 2 – attribute is modified
- 3 – attribute is both read and modified.

Table C:2 Persistent state attributes modified by micro-services for files & collections

| Persistent State Variable Sets | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2 | 3 |
| Number of micro-services      | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 3 | 1 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| COLL_ACCESS_COLL_ID           | 2 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| COLL_ACCESS_TYPE              | 2 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| COLL_ACCESS_USER_ID           | 2 | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| COLL_CREATE_TIME              | 2 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| COLL_ID                       | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| COLL_INHERITANCE              | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| COLL_MODIFY_TIME              | 2 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| COLL_NAME                     | 3 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| COLL_OWNER_NAME               | 2 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| COLL_OWNER_ZONE               | 2 | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| COLL_PARENT_NAME              | 2 | 2 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| DATA_ACCESS_DATA_ID           | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| DATA_ACCESS_TYPE              | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| DATA_ACCESS_USER_ID           | 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| DATA_CHECKSUM                 | 1 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

The sets of persistent state information are listed in table C:2. Each persistent state information set identifies whether a persistent state:

- 1 – attribute is read
- 2 – attribute is modified
- 3 – attribute is both read and modified.

Table C:2 Persistent state attributes modified by micro-services for files & collections
<table>
<thead>
<tr>
<th>Persistent State Variable Sets</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
<th>21</th>
<th>22</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_COLL_ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_COMMENTS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_CREATE_TIME</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_EXPIRY</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_MAP_ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_MODIFY_TIME</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_NAME</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_OWNER_NAME</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_OWNER_ZONE</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_PATH</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_REPL_NUM</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_RESGROUP_NAME</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_RESC_NAME</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_SIZE</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_STATUS</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_TYPE_NAME</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DATA_VERSION</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>META_COLL_ATTR_ID</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_COLL_ATTR_NAME</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_COLL_ATTR_UNITS</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_COLL_ATTR_VALUE</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_COLL_CREATE_TIME</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_COLL_MODIFY_TIME</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_DATA_ATTR_ID</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_DATA_ATTR_NAME</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_DATA_ATTR_UNITS</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_DATA_ATTR_VALUE</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_DATA_CREATE_TIME</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>META_DATA_MODIFY_TIME</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>TOKEN_ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TOKEN_NAME</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TOKEN_NAMESPACE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>USER_GROUP_ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>USER_ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>USER_NAME</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>USER_TYPE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>USER_ZONE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ZONE_NAME</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ZONE_TYPE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table C.3 Additional persistent state attribute sets for operations on files and collections.
<table>
<thead>
<tr>
<th>Persistent State Variable Sets</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA_COMMENTS</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_CREATE_TIME</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_EXPIRY</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_MAP_ID</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_MODIFY_TIME</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_NAME</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_OWNER_NAME</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_OWNER_ZONE</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_PATH</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_REPL_NUM</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_RESP_GROUP_NAME</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_RESP_NAME</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_SIZE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_STATUS</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_TYPE_NAME</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATA_VERSION</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>META_COLL_ATTR_ID</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>META_COLL_ATTR_NAME</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>META_COLL_ATTR_UNITS</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>META_COLL_ATTR_VALUE</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>META_DATA_ATTR_ID</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>META_DATA_ATTR_NAME</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>META_DATA_ATTR_UNITS</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>META_DATA_ATTR_VALUE</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_LIMIT</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_MODIFY_TIME</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_OVER</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_RESC_ID</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_USAGE</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_USAGE_RESP_ID</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_USAGE_USER_ID</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_USER_ID</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESC_ID</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESC_MODIFY_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESC_NAME</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESC_ZONE_NAME</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_MODIFY_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_OWNER_ZONE</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOKEN_ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOKEN_NAME</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOKENNAMESPACE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_GROUP_ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_ID</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_MODIFY_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_NAME</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_TYPE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_ZONE</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZONE_ID</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZONE_MODIFY_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZONE_NAME</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table C:4  Persistent state attributes modified by micro-services for audit trails, rules, and users

<table>
<thead>
<tr>
<th>Persistent State Variable Sets</th>
<th>Number of micro-services</th>
<th>AUDIT_ACTION_ID</th>
<th>AUDIT_COMMENT</th>
<th>AUDIT_CREATE_TIME</th>
<th>AUDIT_MODIFY_TIME</th>
<th>AUDIT_OBJ_ID</th>
<th>AUDIT_USER_ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>DVM_BASE_MAP_BASE_NAME</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_BASE_MAP_CREATE_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_BASE_MAP_MODIFY_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_BASE_MAP_OWNER_NAME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_BASE_MAP_OWNER_ZONE</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_BASE_MAP_VERSION</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_BASE_NAME</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_CONDITION</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_CREATE_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_EXT_VAR_NAME</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_ID</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_INT_MAP_PATH</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_MODIFY_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_OWNER_NAME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_OWNER_ZONE</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DVM_VERSION</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_BASE_MAP_BASE_NAME</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_BASE_MAP_CREATE_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_BASE_MAP_MODIFY_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_BASE_MAP_OWNER_NAME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_BASE_MAP_OWNER_ZONE</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_BASE_MAP_VERSION</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_BASE_NAME</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_CREATE_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_EXT_FUNC_NAME</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_ID</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_INT_FUNC_NAME</td>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_MODIFY_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_OWNER_NAME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FNM_OWNER_ZONE</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>META_COLL_ATTR_ID</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>META_DATA_ATTR_ID</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRVC_MODULE_NAME</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRVC_NAME</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRVC_SIGNATURE</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRVC_VERSION</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRVC_HOST</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRVC_ID</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRVC_LANGUAGE</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRVC_LOCATION</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRVC_STATUS</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSRVC_TYPE_NAME</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_LIMIT</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_MODIFY_TIME</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_OVER</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_RESC_ID</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_USAGE</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_USAGE_RESC_ID</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_USAGE_USER_ID</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QUOTA_USER_ID</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistent State Variable Sets</td>
<td>1</td>
<td>4</td>
<td>9</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>RESC_GROUP_RESC_ID</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESC_GROUP_NAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESC_ID</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESC_NAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESC_ZONE_NAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RESC_VAULT_PATH</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_BASE_MAP_BASE_NAME</td>
<td></td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_BASE_MAP_MODIFY_TIME</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_BASE_MAP_OWNER_NAME</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_BASE_MAP_OWNER_ZONE</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_BASE_MAP_PRIORITY</td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_BASE_MAP_VERSION</td>
<td></td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_BASE_NAME</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_BODY</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_CONDITION</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EVENT</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EXEC_ADDRESS</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EXEC_ESTIMATED_EXE_TIME</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EXEC_FREQUENCY</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EXEC_ID</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EXEC_NAME</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EXEC_NOTIFICATION_ADDR</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EXEC_PRIORITY</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EXEC_REI_FILE_PATH</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EXEC_TIME</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_EXEC_USER_NAME</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_ID</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_NAME</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RULE_RECOVERY</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLD_RESC_NAME</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SLD_CREATE_TIME</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOKEN_ID</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOKEN_NAME</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOKEN_NAMESPACE</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOKEN_VALUE2</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_COMMENT</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_CREATE_TIME</td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_GROUP_ID</td>
<td></td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_ID</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>USER_INFO</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_MODIFY_TIME</td>
<td></td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_NAME</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>USER_TYPE</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>USER_ZONE</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZONE_NAME</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ZONE_TYPE</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Appendix D: Persistent State Variables

The persistent state variables that can be queried are listed below. Note that many of the attributes are maintained and set by the iRODS servers, independently of the micro-services and the policy-enforcement points.

### Table D:1 Persistent State Variables

<table>
<thead>
<tr>
<th>Persistent State Attribute</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUDIT_ACTION_ID</td>
<td>Internal identifier for type of action that is audited</td>
</tr>
<tr>
<td>AUDIT_COMMENT</td>
<td>Comment on audit action for this instance</td>
</tr>
<tr>
<td>AUDIT_CREATE_TIME</td>
<td>Creation timestamp for audit action</td>
</tr>
<tr>
<td>AUDIT_MODIFY_TIME</td>
<td>Modification timestamp for audit action</td>
</tr>
<tr>
<td>AUDIT_OBJ_ID</td>
<td>Internal Identifier of the object (data, collection, user, etc.) on which the audit action was performed</td>
</tr>
<tr>
<td>AUDIT_USER_ID</td>
<td>Internal Identity of user whose action was audited</td>
</tr>
<tr>
<td>COLL_ACCESS_COLL_ID</td>
<td>Aliased Collection identifier used for access control</td>
</tr>
<tr>
<td>COLL_ACCESS_NAME</td>
<td>Access string for collection (cf. DATA_ACCESS_NAME)</td>
</tr>
<tr>
<td>COLL_ACCESS_TYPE</td>
<td>Internal identifier for access name</td>
</tr>
<tr>
<td>COLL_ACCESS_USER_ID</td>
<td>Internal identifier of the user whose action is audited.</td>
</tr>
<tr>
<td>COLL_COMMENTS</td>
<td>Comments about the collection</td>
</tr>
<tr>
<td>COLL_CREATE_TIME</td>
<td>Collection creation timestamp</td>
</tr>
<tr>
<td>COLL_FILEMETA_CREATE_TIME</td>
<td>When a Unix directory is imported into iRODS from client-side, the directory metadata in the file system is captured in the iCAT under COLL_FILEMETA. This is useful when getting the directory back into the client as the “original” metadata can be re-created. The COLL_FILEMETA_CREATE_TIME variable holds the value when the directory metadata was inserted into iCAT</td>
</tr>
<tr>
<td>COLL_FILEMETA_CTIME</td>
<td>Original Unix directory create time at the client-side.</td>
</tr>
<tr>
<td>COLL_FILEMETA_GID</td>
<td>Original Unix Group-id for the directory (used for ACLs) at the client-side.</td>
</tr>
<tr>
<td>COLL_FILEMETA_GROUP</td>
<td>Original Unix Group name for the directory (used for ACLs) at the client-side.</td>
</tr>
<tr>
<td>COLL_FILEMETA_MODE</td>
<td>Original Unix ACL for the directory at the client-side.</td>
</tr>
<tr>
<td>COLL_FILEMETA_MODIFY_TIME</td>
<td>Value when the directory metadata was modified in iCAT</td>
</tr>
<tr>
<td>COLL_FILEMETA_MTIME</td>
<td>Original Unix timestamp for last modification at the client-side</td>
</tr>
<tr>
<td>COLL_FILEMETA_OBJ_ID</td>
<td>Original Unix object id for the directory at the client-side.</td>
</tr>
<tr>
<td>COLL_FILEMETA_OWNER</td>
<td>Original Unix owner for the directory at the client-side.</td>
</tr>
<tr>
<td>COLL_FILEMETA_SOURCE_PATH</td>
<td>Original Unix path for the directory at the client-side.</td>
</tr>
<tr>
<td>COLL_FILEMETA_UID</td>
<td>Original Unix user-id of owner for the directory at the client-side.</td>
</tr>
<tr>
<td>COLL_ID</td>
<td>Collection internal identifier</td>
</tr>
<tr>
<td>COLL_INHERITANCE</td>
<td>Attributes inherited by sub-collections from parent-collection: ACL, metadata, pins, locks</td>
</tr>
<tr>
<td>COLL_MAP_ID</td>
<td>Internal identifier denoting the type of collection.</td>
</tr>
<tr>
<td>COLL_MODIFY_TIME</td>
<td>Last modification timestamp for collection</td>
</tr>
<tr>
<td>COLL_NAME</td>
<td>Logical collection name</td>
</tr>
<tr>
<td>COLL_OWNER_NAME</td>
<td>Collection owner</td>
</tr>
<tr>
<td>COLL_OWNER_ZONE</td>
<td>Home zone of the collection owner</td>
</tr>
<tr>
<td>COLL_PARENT_NAME</td>
<td>Parent collection name</td>
</tr>
<tr>
<td>COLL_TOKEN_NAMESPACE</td>
<td>See TOKEN_NAMESPACE (also DATA_TOKEN_NAMESPACE), not used</td>
</tr>
<tr>
<td>DATA_ACCESS_DATA_ID</td>
<td>Internal identifier of the digital object for which access is defined</td>
</tr>
<tr>
<td>DATA_ACCESS_NAME</td>
<td>Access string in iCAT used for data, collections, etc. (e.g. read object)</td>
</tr>
<tr>
<td>DATA_ACCESS_TYPE</td>
<td>Internal iCAT identifier</td>
</tr>
<tr>
<td>DATA_ACCESS_USER_ID</td>
<td>User or group (name) for which the access is defined on digital object</td>
</tr>
<tr>
<td>DATA_CHECKSUM</td>
<td>Checksum stored as tagged list: &lt;BINHEX&gt;12344&lt;/BINHEX&gt; &lt;MD5&gt;22234422&lt;/MD5&gt;</td>
</tr>
<tr>
<td>DATA_COLL_ID</td>
<td>Collection internal identifier</td>
</tr>
<tr>
<td>DATA_COMMENTS</td>
<td>Comments about the digital object</td>
</tr>
<tr>
<td>DATA_CREATE_TIME</td>
<td>Creation timestamp for the digital object</td>
</tr>
<tr>
<td>DATA_EXPIRY</td>
<td>Expiration date for the digital object</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DATA_FILEMETA_CREATE_TIME</td>
<td>When a Unix file is imported into iRODS from client-side, the file metadata in the file system is captured in the iCAT under DATA_FILEMETA. This is useful when getting the file back into the client as the “original” metadata can be re-created. The DATA_FILEMETA_CREATE_TIME variable holds the value when the file metadata was inserted into iCAT.</td>
</tr>
<tr>
<td>DATA_FILEMETA_CTIME</td>
<td>Original Unix file create time at the client-side.</td>
</tr>
<tr>
<td>DATA_FILEMETA_GID</td>
<td>Original Unix Group id for the file (used for ACLs) at the client-side.</td>
</tr>
<tr>
<td>DATA_FILEMETA_GROUP</td>
<td>Original Unix Group name for the directory file (used for ACLs) at the client-side.</td>
</tr>
<tr>
<td>DATA_FILEMETA_MODE</td>
<td>Original Unix ACL for the file at the client-side.</td>
</tr>
<tr>
<td>DATA_FILEMETA_MODIFY_TIME</td>
<td>Value when the file metadata was modified in iCAT.</td>
</tr>
<tr>
<td>DATA_FILEMETA_MTIME</td>
<td>Original Unix timestamp for last modification at the client-side.</td>
</tr>
<tr>
<td>DATA_FILEMETA_OBJ_ID</td>
<td>Original Unix object id for the file at the client-side.</td>
</tr>
<tr>
<td>DATA_FILEMETA_OWNER</td>
<td>Original Unix owner for the file at the client-side.</td>
</tr>
<tr>
<td>DATA_FILEMETA_SOURCE_PATH</td>
<td>Original Unix path for the file at the client-side.</td>
</tr>
<tr>
<td>DATA_FILEMETA_UID</td>
<td>Original Unix user-id of owner for the file at the client-side.</td>
</tr>
<tr>
<td>DATA_ID</td>
<td>Unique Data internal identifier. A digital object is identified by (zone, collection, data name, replica, version). The identifier is same across replicas and versions.</td>
</tr>
<tr>
<td>DATA_MAP_ID</td>
<td>Internal identifier denoting the type of data.</td>
</tr>
<tr>
<td>DATA_MODIFY_TIME</td>
<td>Last modification timestamp for the digital object.</td>
</tr>
<tr>
<td>DATA_NAME</td>
<td>Logical name of the digital object.</td>
</tr>
<tr>
<td>DATA_OWNER_NAME</td>
<td>User who created the object.</td>
</tr>
<tr>
<td>DATA_OWNER_ZONE</td>
<td>Home zone of the user who created the object.</td>
</tr>
<tr>
<td>DATA_PATH</td>
<td>Physical path name for digital object in resource.</td>
</tr>
<tr>
<td>DATA_REPL_NUM</td>
<td>Replica number starting with “1”.</td>
</tr>
<tr>
<td>DATA_REPL_STATUS</td>
<td>Replica status: locked, is-deleted, pinned, hide.</td>
</tr>
<tr>
<td>DATA_RES_GROUP_NAME</td>
<td>Name of resource group in which data is stored.</td>
</tr>
<tr>
<td>DATA_RES_NAME</td>
<td>Logical name of storage resource.</td>
</tr>
<tr>
<td>DATA_SIZE</td>
<td>Size of the digital object in bytes.</td>
</tr>
<tr>
<td>DATA_STATUS</td>
<td>Digital object status: locked, is-deleted, pinned, hide.</td>
</tr>
<tr>
<td>DATA_TOKEN_NAMESPACE</td>
<td>Namespaces of the data token: e.g. data type, not used.</td>
</tr>
<tr>
<td>DATA_TYPE_NAME</td>
<td>Type of data: jpeg image, PDF document.</td>
</tr>
<tr>
<td>DATA_VERSION</td>
<td>Version string assigned to the digital object. Older versions of replicas have a negative replica number.</td>
</tr>
<tr>
<td>DVM_BASE_MAP_BASE_NAME</td>
<td>Name for the Data Base of Data Variable Set of Maps (e.g. “core” in core.dvm).</td>
</tr>
<tr>
<td>DVM_BASE_MAP_COMMENT</td>
<td>Comments for DVM_BASE_MAP.</td>
</tr>
<tr>
<td>DVM_BASE_MAP_CREATE_TIME</td>
<td>Creation time for DVM_BASE_MAP.</td>
</tr>
<tr>
<td>DVM_BASE_MAP_MODIFY_TIME</td>
<td>Last Modification time for DVM_BASE_MAP.</td>
</tr>
<tr>
<td>DVM_BASE_MAP_OWNER_NAME</td>
<td>Owner’s name of the DVM_BASE_MAP.</td>
</tr>
<tr>
<td>DVM_BASE_MAP_OWNER_ZONE</td>
<td>Owner’s zone name of the DVM_BASE_MAP.</td>
</tr>
<tr>
<td>DVM_BASE_MAP_VERSION</td>
<td>Version of the DVM_BASE_MAP (empty or 0 means current).</td>
</tr>
<tr>
<td>DVM_BASE_NAME</td>
<td>Foreign key reference to DVM_BASE_MAP_BASE_NAME.</td>
</tr>
<tr>
<td>DVM_COMMENT</td>
<td>Comment for the DVM.</td>
</tr>
<tr>
<td>DVM_CONDITION</td>
<td>Condition for applying the DVM Mapping corresponding to DVM_EXT_VAR_NAME.</td>
</tr>
<tr>
<td>DVM_CREATE_TIME</td>
<td>Creation time of the DVM Mapping.</td>
</tr>
<tr>
<td>DVM_EXT_VAR_NAME</td>
<td>External name for the Map (the actual $-variable).</td>
</tr>
<tr>
<td>DVM_ID</td>
<td>An internal identifier for DVM Mapping.</td>
</tr>
<tr>
<td>DVM_INT_MAP_PATH</td>
<td>Internal Structure path in REI corresponding to DVM_EXT_VAR_NAME.</td>
</tr>
<tr>
<td>DVM_MODIFY_TIME</td>
<td>Last modification time for the DVM Mapping.</td>
</tr>
<tr>
<td>DVM_OWNER_NAME</td>
<td>Owner’s name of the DVM Mapping.</td>
</tr>
<tr>
<td>DVM_OWNER_ZONE</td>
<td>Owner’s zone name of the DVM Mapping.</td>
</tr>
<tr>
<td>DVM_STATUS</td>
<td>Status of the DVMMapping (empty is valid).</td>
</tr>
<tr>
<td>DVM_VERSION</td>
<td>Version for the DVM Mapping (empty or 0 means current).</td>
</tr>
<tr>
<td>FNM_BASE_MAP_BASE_NAME</td>
<td>Name for the Data Base of Function Name Set of Maps (e.g. “core” in core.fnm). This can be used for giving virtual names for micro-services and rules and for versioning names for the same.</td>
</tr>
<tr>
<td>FNM_BASE_MAP_COMMENT</td>
<td>Comments for FNM_BASE_MAP.</td>
</tr>
<tr>
<td>FNM_BASE_MAP_CREATE_TIME</td>
<td>Creation time for FNM_BASE_MAP.</td>
</tr>
<tr>
<td>FNM_BASE_MAP_MODIFY_TIME</td>
<td>Last Modification time for FNM_BASE_MAP.</td>
</tr>
<tr>
<td>FNM_BASE_MAP_OWNER_NAME</td>
<td>Owner’s name of the FNM_BASE_MAP.</td>
</tr>
<tr>
<td>FNM_BASE_MAP_OWNER_ZONE</td>
<td>Owner’s zone name of the FNM_BASE_MAP.</td>
</tr>
<tr>
<td>FNM_BASE_MAP_VERSION</td>
<td>Version of the FNM_BASE_MAP (empty or 0 means current).</td>
</tr>
<tr>
<td>FNM_BASE_NAME</td>
<td>Foreign key reference to FNM_BASE_MAP_BASE_NAME.</td>
</tr>
<tr>
<td>FNM_COMMENT</td>
<td>Comment for the FNM Mapping.</td>
</tr>
<tr>
<td>FNM_CREATE_TIME</td>
<td>Creation time of the FNM Mapping.</td>
</tr>
<tr>
<td>FNM_EXT_FUNC_NAME</td>
<td>External name for the FNM Mapping.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FNM_ID</td>
<td>An internal identifier for FNM Mapping</td>
</tr>
<tr>
<td>FNM_INT_FUNC_NAME</td>
<td>Internal Structure path in REI corresponding to FNM_EXT_FUNC_NAME</td>
</tr>
<tr>
<td>FNM_MODIFY_TIME</td>
<td>Last modification time for the FNM Mapping</td>
</tr>
<tr>
<td>FNM_OWNER_NAME</td>
<td>Owner's name of the FNM Mapping</td>
</tr>
<tr>
<td>FNM_OWNER_ZONE</td>
<td>Owner's zone name of the FNM Mapping</td>
</tr>
<tr>
<td>FNM_STATUS</td>
<td>Status of the FNM Mapping (empty is valid)</td>
</tr>
<tr>
<td>FNM_VERSION</td>
<td>Version for the FNM Mapping (empty or 0 means current)</td>
</tr>
<tr>
<td>META_ACCESS_META_ID</td>
<td>Internal identifier of the (AVU) metadata for which access is defined</td>
</tr>
<tr>
<td>META_ACCESS_NAME</td>
<td>See DATA_ACCESS_NAME</td>
</tr>
<tr>
<td>META_ACCESS_TYPE</td>
<td>Internal ICAT identifier</td>
</tr>
<tr>
<td>META_ACCESS_USER_ID</td>
<td>User or group (name) for which the access is defined on metadata</td>
</tr>
<tr>
<td>META_COLL_ATTR_ID</td>
<td>Internal identifier for metadata attribute for collection</td>
</tr>
<tr>
<td>META_COLL_ATTR_NAME</td>
<td>Metadata attribute name for collection</td>
</tr>
<tr>
<td>META_COLL_ATTR_UNITS</td>
<td>Metadata attribute units for collection</td>
</tr>
<tr>
<td>META_COLL_ATTR_VALUE</td>
<td>Metadata attribute value for collection</td>
</tr>
<tr>
<td>META_COLL_CREATE_TIME</td>
<td>Creation time for the metadata for collections</td>
</tr>
<tr>
<td>META_COLL_MODIFY_TIME</td>
<td>Last modification time for the metadata for collections</td>
</tr>
<tr>
<td>META_DATA_ATTR_ID</td>
<td>Internal identifier for metadata attribute for digital object</td>
</tr>
<tr>
<td>META_DATA_ATTR_NAME</td>
<td>Metadata attribute name for digital object</td>
</tr>
<tr>
<td>META_DATA_ATTR_UNITS</td>
<td>Metadata attribute units for digital object</td>
</tr>
<tr>
<td>META_DATA_ATTR_VALUE</td>
<td>Metadata attribute value for digital object</td>
</tr>
<tr>
<td>META_DATA_CREATE_TIME</td>
<td>Time stamp when metadata was created</td>
</tr>
<tr>
<td>META_DATA_MODIFY_TIME</td>
<td>Time stamp when metadata was modified</td>
</tr>
<tr>
<td>META_MET2_ATTR_ID</td>
<td>Internal identifier for metadata attribute for metadata</td>
</tr>
<tr>
<td>META_MET2_ATTR_NAME</td>
<td>Metadata attribute name for metadata</td>
</tr>
<tr>
<td>META_MET2_ATTR_UNITS</td>
<td>Metadata attribute units for metadata</td>
</tr>
<tr>
<td>META_MET2_ATTR_VALUE</td>
<td>Metadata attribute value for metadata</td>
</tr>
<tr>
<td>META_MET2_CREATE_TIME</td>
<td>Creation time for the metadata for metadata</td>
</tr>
<tr>
<td>META_MET2_MODIFY_TIME</td>
<td>Last modification time for the metadata for metadata</td>
</tr>
<tr>
<td>META_MSRVC_ATTR_ID</td>
<td>Internal identifier for metadata attribute for micro-service</td>
</tr>
<tr>
<td>META_MSRVC_ATTR_NAME</td>
<td>Metadata attribute name for micro-service</td>
</tr>
<tr>
<td>META_MSRVC_ATTR_UNITS</td>
<td>Metadata attribute units for micro-service</td>
</tr>
<tr>
<td>META_MSRVC_ATTR_VALUE</td>
<td>Metadata attribute value for micro-service</td>
</tr>
<tr>
<td>META_MSRVC_CREATE_TIME</td>
<td>Creation time for the metadata for micro-service</td>
</tr>
<tr>
<td>META_MSRVC_MODIFY_TIME</td>
<td>Last modification time for the metadata for micro-service</td>
</tr>
<tr>
<td>META_NAMESPACE_COLL</td>
<td>Namespace of collection AVU-triplet attribute</td>
</tr>
<tr>
<td>META_NAMESPACE_DATA</td>
<td>Namespace of digital object AVU-triplet attribute</td>
</tr>
<tr>
<td>META_NAMESPACE_MET2</td>
<td>Namespace of metadata AVU-triplet attribute</td>
</tr>
<tr>
<td>META_NAMESPACE_MSRVC</td>
<td>Namespace of micro-service AVU-triplet attribute</td>
</tr>
<tr>
<td>META_NAMESPACE_RESC</td>
<td>Namespace of resource AVU-triplet attribute</td>
</tr>
<tr>
<td>META_NAMESPACE_RESC_GROUP</td>
<td>Namespace of resource-group AVU-triplet attribute</td>
</tr>
<tr>
<td>META_NAMESPACE_RULE</td>
<td>Namespace of rule AVU-triplet attribute</td>
</tr>
<tr>
<td>META_NAMESPACE_USER</td>
<td>Namespace of user AVU-triplet attribute</td>
</tr>
<tr>
<td>META_RESC_ATTR_ID</td>
<td>Internal identifier for metadata attribute for resource</td>
</tr>
<tr>
<td>META_RESC_ATTR_NAME</td>
<td>Metadata attribute name for resource</td>
</tr>
<tr>
<td>META_RESC_ATTR_UNITS</td>
<td>Metadata attribute units for resource</td>
</tr>
<tr>
<td>META_RESC_ATTR_VALUE</td>
<td>Metadata attribute value for resource</td>
</tr>
<tr>
<td>META_RESC_CREATE_TIME</td>
<td>Creation time for the metadata for resource</td>
</tr>
<tr>
<td>META_RESC_MODIFY_TIME</td>
<td>Last modification time for the metadata for resource</td>
</tr>
<tr>
<td>META_RESC_GROUP_ATTR_ID</td>
<td>Internal identifier for metadata attribute for resource group</td>
</tr>
<tr>
<td>META_RESC_GROUP_ATTR_NAME</td>
<td>Metadata attribute name for resource group</td>
</tr>
<tr>
<td>META_RESC_GROUP_ATTR_UNITS</td>
<td>Metadata attribute units for resource group</td>
</tr>
<tr>
<td>META_RESC_GROUP_ATTR_VALUE</td>
<td>Metadata attribute value for resource group</td>
</tr>
<tr>
<td>META_RESC_GROUP_CREATE_TIME</td>
<td>Creation time for the metadata for resource group</td>
</tr>
<tr>
<td>META_RESC_GROUP_MODIFY_TIME</td>
<td>Last modification time for the metadata for resource group</td>
</tr>
<tr>
<td>META_RULE_ATTR_ID</td>
<td>Internal identifier for metadata attribute for a rule</td>
</tr>
<tr>
<td>META_RULE_ATTR_NAME</td>
<td>Metadata attribute name for a rule</td>
</tr>
<tr>
<td>META_RULE_ATTR_UNITS</td>
<td>Metadata attribute units for a rule</td>
</tr>
<tr>
<td>META_RULE_ATTR_VALUE</td>
<td>Metadata attribute value for a rule</td>
</tr>
<tr>
<td>META_RULE_CREATE_TIME</td>
<td>Creation time for the metadata entry for a rule</td>
</tr>
<tr>
<td>META_RULE_MODIFY_TIME</td>
<td>Last modification time for the metadata for a rule</td>
</tr>
<tr>
<td>META_TOKEN_NAMESPACE</td>
<td>See TOKEN_NAMESPACE</td>
</tr>
<tr>
<td>META_USER_ATTR_ID</td>
<td>Internal identifier for metadata attribute for user</td>
</tr>
<tr>
<td>META_USER_ATTR_NAME</td>
<td>Metadata attribute name for user</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>META_USER_ATTR_UNITS</td>
<td>Metadata attribute units for user</td>
</tr>
<tr>
<td>META_USER_ATTR_VALUE</td>
<td>Metadata attribute value for user</td>
</tr>
<tr>
<td>META_USER_CREATE_TIME</td>
<td>Internal identifier of the (AVU) metadata for which access is defined</td>
</tr>
<tr>
<td>MSRVC_ACCESS_NAME</td>
<td>User or group (name) for which the access is defined on metadata</td>
</tr>
<tr>
<td>MSRVC_ACCESS_TYPE</td>
<td>Internal ICAT identifier</td>
</tr>
<tr>
<td>MSRVC_ACCESS_USER_ID</td>
<td>User or group (name) for which the access is defined on the micro-service</td>
</tr>
<tr>
<td>MSRVC_COMMENT</td>
<td>Comments for micro-service</td>
</tr>
<tr>
<td>MSRVC_CREATE_TIME</td>
<td>Creation time for the micro-service</td>
</tr>
<tr>
<td>MSRVC_DOXYGEN</td>
<td>Doxygen documentation for the micro-service</td>
</tr>
<tr>
<td>MSRVC_HOST</td>
<td>Host types at which the micro-service can be executed</td>
</tr>
<tr>
<td>MSRVC_ID</td>
<td>Internal Id for the micro-service</td>
</tr>
<tr>
<td>MSRVC_LANGUAGE</td>
<td>Language in which the micro-service is written</td>
</tr>
<tr>
<td>MSRVC_LOCATION</td>
<td>The Location of the micro-service executable</td>
</tr>
<tr>
<td>MSRVC_MODIFY_TIME</td>
<td>Last Modification time for the micro-service</td>
</tr>
<tr>
<td>MSRVC_MODULE_NAME</td>
<td>Module name for the micro-service</td>
</tr>
<tr>
<td>MSRVC_OWNER_NAME</td>
<td>Owner name of the micro-service</td>
</tr>
<tr>
<td>MSRVC_OWNER_ZONE</td>
<td>Owner’s zone name of the micro-service</td>
</tr>
<tr>
<td>MSRVC_SIGNATURE</td>
<td>Digital signature (checksum) for the micro-service</td>
</tr>
<tr>
<td>MSRVC_STATUS</td>
<td>Status of the micro-service</td>
</tr>
<tr>
<td>MSRVC_TOKEN_NAMESPACE</td>
<td>See TOKEN_NAMESPACE</td>
</tr>
<tr>
<td>MSRVC_TYPE_NAME</td>
<td>Type of the micro-service</td>
</tr>
<tr>
<td>MSRVC_VARIATIONS</td>
<td>Variations (or forms) of the micro-service</td>
</tr>
<tr>
<td>MSRVC_VER_COMMENT</td>
<td>Comments on the micro-service</td>
</tr>
<tr>
<td>MSRVC_VER_CREATE_TIME</td>
<td>Creation time of version of the micro-service</td>
</tr>
<tr>
<td>MSRVC_VER_MODIFY_TIME</td>
<td>Last modification time of version of the micro-service</td>
</tr>
<tr>
<td>MSRVC_VER_OWNER_NAME</td>
<td>Owner name of the version of the micro-service</td>
</tr>
<tr>
<td>MSRVC_VER_OWNER_ZONE</td>
<td>Owner zone name of the version of the micro-service</td>
</tr>
<tr>
<td>MSRVC_VERSION</td>
<td>Version of the micro-service</td>
</tr>
<tr>
<td>QUOTA_LIMIT</td>
<td>High limit for quota for resource in QUOTA_RES_ID for QUOTA_USER_ID</td>
</tr>
<tr>
<td>QUOTA_MODIFY_TIME</td>
<td>Last modification time of quota</td>
</tr>
<tr>
<td>QUOTA_OVER</td>
<td>Flag if quota is exceeded</td>
</tr>
<tr>
<td>QUOTA_RES_ID</td>
<td>Internal Resource ID for quota</td>
</tr>
<tr>
<td>QUOTA_RES_NAME</td>
<td>Resource Name for quota</td>
</tr>
<tr>
<td>QUOTA_USAGE</td>
<td>Name of Usage for quota (normally write)</td>
</tr>
<tr>
<td>QUOTA_USAGE_MODIFY_TIME</td>
<td>Last modification time of quota usage</td>
</tr>
<tr>
<td>QUOTA_USAGE_RES_ID</td>
<td>Internal Resource ID for quota usage</td>
</tr>
<tr>
<td>QUOTA USAGE_USER_ID</td>
<td>Internal User ID for quota usage</td>
</tr>
<tr>
<td>QUOTA_USER_ID</td>
<td>Internal User ID for quota</td>
</tr>
<tr>
<td>QUOTA_USER_NAME</td>
<td>User Name for Quota</td>
</tr>
<tr>
<td>QUOTA_USER_TYPE</td>
<td>User type name for quota</td>
</tr>
<tr>
<td>QUOTA_USER_ZONE</td>
<td>User zone name for quota</td>
</tr>
<tr>
<td>RESC_ACCESS_RES_ID</td>
<td>Internal identifier of the resource for which access is defined</td>
</tr>
<tr>
<td>RESC_ACCESS_TYPE</td>
<td>Internal ICAT identifier</td>
</tr>
<tr>
<td>RESC_ACCESS_USER_ID</td>
<td>User or group (name) for which the access is defined on resource</td>
</tr>
<tr>
<td>RESC_CLASS_NAME</td>
<td>Resource class: primary, secondary, archival</td>
</tr>
<tr>
<td>RESC_COMMENT</td>
<td>Comment about resource</td>
</tr>
<tr>
<td>RESC_CREATE_TIME</td>
<td>Creation timestamp of resource</td>
</tr>
<tr>
<td>RESC_FREE_SPACE</td>
<td>Free space available on resource</td>
</tr>
<tr>
<td>RESC_FREE_SPACE_TIME</td>
<td>Time at which free space was computed</td>
</tr>
<tr>
<td>RESC_GROUP_ID</td>
<td>Internal Id for resource group</td>
</tr>
<tr>
<td>RESC_GROUP_NAME</td>
<td>Logical name of the resource group</td>
</tr>
<tr>
<td>RESC_GROUP_RESID</td>
<td>Internal identifier for the resource group</td>
</tr>
<tr>
<td>RESC_ID</td>
<td>Internal resource identifier for resource in the group</td>
</tr>
<tr>
<td>RESC_INFO</td>
<td>Tagged information list: &lt;MAX_OBJ_SIZE&gt; 2GBB &lt;/MAX_OBJ_SIZE&gt;</td>
</tr>
<tr>
<td>RESC_LOC</td>
<td>Resource IP address</td>
</tr>
<tr>
<td>RESC_MODIFY_TIME</td>
<td>Last modification timestamp for resource</td>
</tr>
<tr>
<td>RESC_NAME</td>
<td>Logical name of the resource</td>
</tr>
<tr>
<td>RESC_STATUS</td>
<td>Operational status of resource</td>
</tr>
<tr>
<td>RESC_TOKEN_NAMESPACE</td>
<td>See TOKEN_NAMESPACE</td>
</tr>
<tr>
<td>RESC_TYPE_NAME</td>
<td>Resource type: HPSS, SamFS, database, or b</td>
</tr>
</tbody>
</table>

234
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESC_VAULT_PATH</td>
<td>Resource path for storing files</td>
</tr>
<tr>
<td>RESC_ZONE_NAME</td>
<td>Name of the iCAT, unique globally</td>
</tr>
<tr>
<td>RULE_ACCESS_NAME</td>
<td>Internal identifier of the iRODS rule for which access is defined</td>
</tr>
<tr>
<td>RULE_ACCESS_RULE_ID</td>
<td>See DATA_ACCESS_NAME</td>
</tr>
<tr>
<td>RULE_ACCESS_TYPE</td>
<td>Internal ICAT identifier</td>
</tr>
<tr>
<td>RULE_ACCESS_USER_ID</td>
<td>User or group (name) for which the access is defined on iRODS rule</td>
</tr>
<tr>
<td>RULE_BASE_MAP_BASE_NAME</td>
<td>Name for the Data Base of Rule Set of Maps (e.g., “core” in core.re)</td>
</tr>
<tr>
<td>RULE_BASE_MAP_COMMENT</td>
<td>Comments for RULE_BASE_MAP</td>
</tr>
<tr>
<td>RULE_BASE_MAP_CREATE_TIME</td>
<td>Creation time for RULE_BASE_MAP</td>
</tr>
<tr>
<td>RULE_BASE_MAP_MODIFY_TIME</td>
<td>Last Modification time for RULE_BASE_MAP</td>
</tr>
<tr>
<td>RULE_BASE_MAP_OWNER_NAME</td>
<td>Owner’s name of the RULE_BASE_MAP</td>
</tr>
<tr>
<td>RULE_BASE_MAP_OWNER_ZONE</td>
<td>Owner’s zone name of the RULE_BASE_MAP</td>
</tr>
<tr>
<td>RULE_BASE_MAP_PRIORITY</td>
<td>Prioritization of the RULE_BASE_MAP (empty or 0 means current). This tells which map has priority over other maps. This can define a tree/forest.</td>
</tr>
<tr>
<td>RULE_BASE_MAP_VERSION</td>
<td>Version of the RULE_BASE_MAP (empty or 0 means current)</td>
</tr>
<tr>
<td>RULE_BASE_NAME</td>
<td>Rule base to which the rule is a member</td>
</tr>
<tr>
<td>RULE_BODY</td>
<td>Body of the rule</td>
</tr>
<tr>
<td>RULE_COMMENT</td>
<td>Comments on the rule</td>
</tr>
<tr>
<td>RULE_CONDITION</td>
<td>Condition of the rule</td>
</tr>
<tr>
<td>RULE_CREATE_TIME</td>
<td>Creation time of the rule</td>
</tr>
<tr>
<td>RULE_DESCR_1</td>
<td>Description of rule (1)</td>
</tr>
<tr>
<td>RULE_DESCR_2</td>
<td>Description of rule (2)</td>
</tr>
<tr>
<td>RULE_DOLLAR_VARS</td>
<td>Session variables used in the rule</td>
</tr>
<tr>
<td>RULE_EVENT</td>
<td>Event name of the rule (can be viewed as rule name)</td>
</tr>
<tr>
<td>RULE_EXEC_ADDRESS</td>
<td>Host name where the delayed Rule will be executed</td>
</tr>
<tr>
<td>RULE_EXEC_ESTIMATED_EXE_TIME</td>
<td>Estimated execution time for the delayed Rule</td>
</tr>
<tr>
<td>RULE_EXEC_FREQUENCY</td>
<td>Delayed Rule execution frequency</td>
</tr>
<tr>
<td>RULE_EXEC_ID</td>
<td>Internal identifier for a delayed Rule execution request</td>
</tr>
<tr>
<td>RULE_EXEC_LAST_EXE_TIME</td>
<td>Previous execution time for the delayed Rule</td>
</tr>
<tr>
<td>RULE_EXEC_NAME</td>
<td>Logical name for a delayed Rule execution request</td>
</tr>
<tr>
<td>RULE_EXEC_NOTIFICATION_ADDR</td>
<td>Notification address for delayed Rule completion</td>
</tr>
<tr>
<td>RULE_EXEC_PRIORITY</td>
<td>Delayed Rule execution priority</td>
</tr>
<tr>
<td>RULE_EXEC_REI_FILE_PATH</td>
<td>Path of the file where the context (REI) of the delayed Rule is stored</td>
</tr>
<tr>
<td>RULE_EXEC_STATUS</td>
<td>Current status of the delayed Rule</td>
</tr>
<tr>
<td>RULE_EXEC_TIME</td>
<td>Time when the delayed Rule will be executed</td>
</tr>
<tr>
<td>RULE_EXEC_USER_NAME</td>
<td>User requesting a delayed Rule execution</td>
</tr>
<tr>
<td>RULE_ICAT_ELEMENTS</td>
<td>Permanent (#-variables) affected by the rule</td>
</tr>
<tr>
<td>RULE_ID</td>
<td>Internal identifier for the rule</td>
</tr>
<tr>
<td>RULE_INPUT_PARAMS</td>
<td>Parameters used as input when invoking the rule</td>
</tr>
<tr>
<td>RULE_MODIFY_TIME</td>
<td>Last modification time of the rule</td>
</tr>
<tr>
<td>RULE_NAME</td>
<td>Name of the rule (can be different from RULE_EVENT)</td>
</tr>
<tr>
<td>RULE_OUTPUT_PARAMS</td>
<td>Output parameters set by the rule invocation</td>
</tr>
<tr>
<td>RULE_OWNER_NAME</td>
<td>Owner name of the rule</td>
</tr>
<tr>
<td>RULE_OWNER_ZONE</td>
<td>Owner's zone name of the rule</td>
</tr>
<tr>
<td>RULE_RECOVERY</td>
<td>Recovery part of the rule</td>
</tr>
<tr>
<td>RULE_SIDE_EFFECTS</td>
<td>Side effects (%-variables) – used as a semantic of what the rule does</td>
</tr>
<tr>
<td>RULE_STATUS</td>
<td>Status of the rule (valid/active or otherwise)</td>
</tr>
<tr>
<td>RULE_TOKEN_NAMESPACE</td>
<td>See TOKEN_NAMESPACE</td>
</tr>
<tr>
<td>RULE_VERSION</td>
<td>Version of the rule</td>
</tr>
<tr>
<td>SL_CPU_USED</td>
<td>Server load information: cpu used. Server load information is computed periodically for all servers in the grid, if enabled by the administrator.</td>
</tr>
<tr>
<td>SL_CREATE_TIME</td>
<td>Server load information: creation time of the entry</td>
</tr>
<tr>
<td>SL_DISK_SPACE</td>
<td>Server load information: disk space used</td>
</tr>
<tr>
<td>SL_HOST_NAME</td>
<td>Server load information: host name of the server</td>
</tr>
<tr>
<td>SL_MEM_USED</td>
<td>Server load information: memory used</td>
</tr>
<tr>
<td>SL_NET_INPUT</td>
<td>Server load information: network input load</td>
</tr>
<tr>
<td>SL_NET_OUTPUT</td>
<td>Server load information: network output load</td>
</tr>
<tr>
<td>SL_RES_NAME</td>
<td>Server load information: resource for which disk space is provided</td>
</tr>
<tr>
<td>SL_RUNQ_LOAD</td>
<td>Server load information: run queue load</td>
</tr>
<tr>
<td>SL_SWAP_USED</td>
<td>Server load information: swap space used</td>
</tr>
<tr>
<td>SLD_CREATE_TIME</td>
<td>Server load digest information: digest creation time</td>
</tr>
<tr>
<td>SLD_LOAD_FACTOR</td>
<td>Server load information: load factor computed rom server load information</td>
</tr>
<tr>
<td>SLD_RES_NAME</td>
<td>Server load information: resource name for which the load factor is computed</td>
</tr>
<tr>
<td>TICKET_ALLOWED_GROUP_NAME</td>
<td>User group to which the ticket (TICKET_ALLOWED_GROUP_TICKET_ID) is valid</td>
</tr>
<tr>
<td>TICKET_ALLOWED_GROUP_TICKET_ID</td>
<td>Identifier for the ticket</td>
</tr>
<tr>
<td>TICKET_ALLOWED_HOST</td>
<td>Host for which the ticket (TICKET_ALLOWED_HOST_TICKET_ID) is valid. Allows invocation of the ticket-based access only from this host. Useful for scheduled jobs.</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>TICKET_ALLOWED_HOST_TICKET_ID</td>
<td>Identifier for the ticket</td>
</tr>
<tr>
<td>TICKET_ALLOWED_USER_NAME</td>
<td>User to which the ticket (TICKET_ALLOWED_GROUP_TICKET_ID) is valid</td>
</tr>
<tr>
<td>TICKET_ALLOWED_USER_TICKET_ID</td>
<td>Identifier for the ticket</td>
</tr>
<tr>
<td>TICKET_COLL_NAME</td>
<td>Collection name on which the ticket is issued</td>
</tr>
<tr>
<td>TICKET_CREATE_TIME</td>
<td>Ticket creation time</td>
</tr>
<tr>
<td>TICKET_DATA_COLL_NAME</td>
<td>Collection name of the object on which the ticket is issued</td>
</tr>
<tr>
<td>TICKET_DATA_NAME</td>
<td>Data name of the object on which the ticket is issued</td>
</tr>
<tr>
<td>TICKET_EXPIRY</td>
<td>Expiration date for a ticket</td>
</tr>
<tr>
<td>TICKET_ID</td>
<td>Identifier for the ticket</td>
</tr>
<tr>
<td>TICKET_MODIFY_TIME</td>
<td>Last modification time for the ticket</td>
</tr>
<tr>
<td>TICKET_OBJECT_ID</td>
<td>(Internal) Object ID for the object on which the ticket is issued</td>
</tr>
<tr>
<td>TICKET_OBJECT_TYPE</td>
<td>Ticket may be for data, resource, user, rule, metadata, zone, collection, token</td>
</tr>
<tr>
<td>TICKET_OWNER_NAME</td>
<td>Name of the person who created the ticket</td>
</tr>
<tr>
<td>TICKET_OWNER_ZONE</td>
<td>Home zone of the person who created the ticket</td>
</tr>
<tr>
<td>TICKET_STRING</td>
<td>Human readable name for the ticket</td>
</tr>
<tr>
<td>TICKET_TYPE</td>
<td>Type of ticket, either &quot;read&quot; or &quot;write&quot;</td>
</tr>
<tr>
<td>TICKET_USER_ID</td>
<td>Identifier of the person who is using the ticket</td>
</tr>
<tr>
<td>TICKET_USES_COUNT</td>
<td>Number of times a ticket has been used</td>
</tr>
<tr>
<td>TICKET_USES_LIMIT</td>
<td>Maximum number of times a ticket may be used</td>
</tr>
<tr>
<td>TICKET_WRITE_BYTE_COUNT</td>
<td>Number of bytes written for accesses through a given ticket</td>
</tr>
<tr>
<td>TICKET_WRITE_BYTE_LIMIT</td>
<td>Maximum number of bytes that may be written using a given ticket</td>
</tr>
<tr>
<td>TICKET_WRITE_FILE_COUNT</td>
<td>Number of files written for accesses through a given ticket</td>
</tr>
<tr>
<td>TICKET_WRITE_FILE_LIMIT</td>
<td>Maximum number of files that can be written using a given ticket</td>
</tr>
<tr>
<td>TOKEN_COMMENT</td>
<td>Comment on token</td>
</tr>
<tr>
<td>TOKEN_ID</td>
<td>Internal identifier for token name</td>
</tr>
<tr>
<td>TOKEN_NAME</td>
<td>A value in the token namespace; e.g. &quot;jpg image&quot;</td>
</tr>
<tr>
<td>TOKEN_NAMESPACE</td>
<td>Namespace for tokens; e.g. data type, resource type, rule type, ...</td>
</tr>
<tr>
<td>TOKEN_VALUE</td>
<td>Additional token information string (e.g. dot extensions for jpg: jpg, .jpg2, .jg)</td>
</tr>
<tr>
<td>TOKEN_VALUE2</td>
<td>Additional token information string</td>
</tr>
<tr>
<td>TOKEN_VALUE3</td>
<td>Additional token information string</td>
</tr>
<tr>
<td>USER_COMMENT</td>
<td>Comment about the user</td>
</tr>
<tr>
<td>USER_CREATE_TIME</td>
<td>Creation timestamp</td>
</tr>
<tr>
<td>USER_DN</td>
<td>Distinguished name in tagged list: &lt;authType&gt;distinguishedName&lt;/authType&gt;</td>
</tr>
<tr>
<td>USER_GROUP_ID</td>
<td>Internal identifier for the user group</td>
</tr>
<tr>
<td>USER_GROUP_NAME</td>
<td>Logical name for the user group</td>
</tr>
<tr>
<td>USER_ID</td>
<td>User internal identifier</td>
</tr>
<tr>
<td>USER_INFO</td>
<td>Tagged information: &lt;EMAIL&gt;<a href="mailto:user@unc.edu">user@unc.edu</a>&lt;/EMAIL&gt; &lt;PHONE&gt;5555555555&lt;/PHONE&gt;</td>
</tr>
<tr>
<td>USER_MODIFY_TIME</td>
<td>Last modification timestamp</td>
</tr>
<tr>
<td>USER_NAME</td>
<td>User name</td>
</tr>
<tr>
<td>USER_TYPE</td>
<td>User role (rodsgroup, rodsadmin, rodsuser, domainadmin, groupadmin, storageadmin, rodscurators)</td>
</tr>
<tr>
<td>USER_ZONE</td>
<td>Home Data Grid or user</td>
</tr>
<tr>
<td>ZONE_COMMENT</td>
<td>Comment about the zone</td>
</tr>
<tr>
<td>ZONE_CONNECTION</td>
<td>Connection information in tagged list: &lt;PASSWORD&gt;RPS1&lt;/PASSWORD&gt; &lt;GSI&gt;DISTNAME&lt;/GSI&gt;</td>
</tr>
<tr>
<td>ZONE_CREATE_TIME</td>
<td>Date and time stamp for creation of a data grid</td>
</tr>
<tr>
<td>ZONE_ID</td>
<td>Data Grid or zone identifier</td>
</tr>
<tr>
<td>ZONE_MODIFY_TIME</td>
<td>Date and time stamp for modification of a data grid</td>
</tr>
<tr>
<td>ZONE_NAME</td>
<td>Data Grid or zone name, name of the iCAT</td>
</tr>
<tr>
<td>ZONE_TYPE</td>
<td>Type of zone: local/remote/other</td>
</tr>
</tbody>
</table>
Appendix E: Protected Data Requirements

The data management requirements are abstracted from the document, https://www.med.unc.edu/security/hipaa/documents/ADMIN0082%20Info%20Security.pdf. Each requirement has been evaluated for the feasibility of creating a computer actionable policy that automates enforcement.

- Document the policies
- Protect the confidentiality, integrity, and availability of information from accidental or intentional unauthorized modification, destruction or disclosure
- Periodic risk assessment to document types of threats and vulnerabilities, and evaluate information assets and technology for data collection, storage, dissemination, and protection
- Protected assets include:
  - Payment card account numbers, card holder name, expiration date, service code, and CID/PINs
  - Legally covered entities
  - Social Security Numbers and personal information
  - Protected Health Information – demographic, physical or mental health, provision of health care, health care payment that identifies the individual
- Protection tasks
  - Data available on demand by an authorized person
  - Data not accessible by unauthorized person or process
  - Encryption
  - Integrity
  - Identify involved person identification
  - Identify involved computer systems
- Security Office
  - Monitor policy distribution to resources
  - Basic security support (accounts, access controls, OS upgrades)
  - Classification of computer resources
  - System design for security controls
  - Vulnerability detection, notification
  - Detection of unauthorized access (audit trails)
  - Training
  - Security audits
  - Reports
- Collection owners
  - Presence of HIPAA information
  - Data retention period
  - Application of policies and procedures for data protection
  - Authorizing access
  - Specifying controls, setting control policies
  - Reporting loss or misuse
• Correcting problems
• Training
• Tracking approval processes for systems

• Data grid administrator – custodian
  o Provide physical safeguards – one-time passwords to access iCAT
  o Provide procedures for security
  o Control access to information
  o Release information through privacy procedures
  o Evaluate cost effectiveness of controls
  o Maintain policies and procedures
  o Promote education
  o Report loss or misuse
  o Respond to security incidents

• User management – projects
  o Review and approve requests for access
  o Update employees’ security records with position and job function changes
  o Update access on employee termination or transfer
  o Revoke physical access to terminated employees
  o Promote training
  o Report loss or misuse
  o Initiate corrective actions
  o Follow recommendations for purchase and implementation of systems

• User
  o Only access information for authorized job responsibilities
  o Comply with access controls
  o Report disclosures of PHI other than for treatment, payment, or health care
  o Keep personal authentication information confidential
  o Report loss or misuse
  o Initiate corrective actions

• Classify information
  o Protected health information
  o Confidential information – PCI, PI
  o Internal information – all information not PHI, Confidential, or Public
  o Public information

• Computer and information control
  o Ownership, licensing of software
  o Inventory of software and computers, users, managers
  o Virus protection, scan all files
  o Access controls
    • authorization by supervisor context based – ticket
    • authorization role based
    • authorization user based
    • authentication – unique user ID
- Controlled passwords
- Biometric
- Tokens in conjunction with a PIN

  **Password security**
  - No re-use or multiple use
  - Minimum length, expiration, encryption during transmission, storage
  - Log unsuccessful attempts
  - Procedures for validating users who request password reset

  - Automatic timeout after period of inactivity
  - Log-off

- Data integrity
  - Transaction audit
  - Replication
  - Checksums
  - Encryption in storage
  - Digital signatures
  - Data validation on entry

- Transmission security
  - Integrity – checksums
  - Encryption in messaging systems

- Remote access
  - Only approved methods and pathways

- Physical access
  - Access controlled areas, HVAC
  - Authentication to data grid and access controls
  - Authentication to workstation, automatic screen savers

- Facility access controls
  - Contingency for emergency operations after disaster
  - Facility security plan – policies and procedures
  - Documented procedures to validate access
  - Documented maintenance of facility

- Emergency access
  - Procedures for authorization, implementation, revocation

- Equipment and media controls
  - Media disposal
  - Track custody of media
  - Data backup

- Other media controls
  - Encryption for storage on removable media
  - Encryption, power-on passwords, auto logoff for mobile devices
  - Ownership of media for assigning responsibility

- Data transfer/printing
  - Approval for bulk download
- De-identification of data – Bitcuator
- Encrypt data transfers

- Social Media
  - No PHI, confidential, or proprietary information
  - No patient identification information
  - No patient photographs

- Audit controls
  - Record activity by users and system administrators
  - Review activity logs
  - Preserve reviews for 6 years

- Evaluation
  - Verify procedures after each operational or environmental change

- Contingency plan
  - Enable recovery of data
    - Document data backup plan
    - Backup data off site
    - Manage access controls on replicas
  - Disaster recovery plan – procedure for restoring data
  - Emergency operation plan – for natural disasters
  - Procedures for testing contingency plans on revision
  - Identify critical components

- Password controls
  - No sharing of passwords
  - Single sign-on system for passwords
  - No passwords on PC
  - No dictionary words
  - Encrypt passwords
  - Maximum of 5 invalid passwords causes lockout for 30 minutes
  - Contain 1 upper case, 1 lower case, 1 number
  - Minimum length of 10 characters
  - Passwords changed annually
  - Maintain history of prior 6 passwords, prevent re-use

- Peer-to-peer
  - F2P file-sharing programs are prohibited
  - Internet storage may not be used for PHI and confidential information
Appendix F: Mauna Loa Sensor Data DMP

Types of Data Produced
Air samples at Mauna Loa Observatory will be collected continuously from air intakes located at five towers – a central tower and four towers located at compass quadrants. Raw data files will contain continuously measured CO₂ concentrations, calibration standards, references standards, daily check standards, and blanks. The sample lines located at compass quadrants were used to examine the influence of source effects associated with wind directions [3,4]. In addition to the CO₂ data, we will record weather data (wind speed and direction, temperature, humidity, precipitation, and cloud cover). Site conditions at Mauna Loa Observatory will also be noted and retained.

The final data product will consist of 5-minute, 15-minute, hourly, daily, and monthly average atmospheric concentration of CO₂ in mole fraction in water-vapor-free air measured at the Mauna Loa Observatory, Hawaii. Data are reported as a dry mole fraction defined as the number of molecules of CO₂ divided by the number of molecules of dry air multiplied by one million (ppm).

The final data product has been thoroughly documented in the open literature [2] and in Scripps Institution of Oceanography Internal Reports [1]. The data generated (raw CO₂ measurements, meteorological data, calibration and reference standards) will be placed in comma-separated-values in plain ASCII format, which are readable over long time periods. The final data file will contain dates for each observation (time, day, month and year) and the average CO₂ concentration. The final data product distributed to most users will occupy less than 500 KB; raw and ancillary data, which will be distributed on request comprise less than 10 MB.

Data and Metadata Standards
Metadata will be comprised of two formats – contextual information about the data in a text based document and ISO 19115 standard metadata in an xml file. These two formats for metadata were chosen to provide a full explanation of the data (text format) and to ensure compatibility with international standards (xml format). The standard XML file will be more complete; the document file will be a human-readable summary of the XML file.

Policies for Access and Sharing
The final data product will be released to the public as soon as the recalibration of standard gases has been completed and the data have been prepared, typically within six months of collection. There is no period of exclusive use by the data collectors. Users can access documentation and final monthly CO₂ data files via the Scripps CO₂ Program website (http://scrippsc02.ucsd.edu). The data will be made available via ftp download from the Scripps Institution of Oceanography Computer Center. Raw data (continuous concentration measurements, weather data, etc.) will be maintained on an internally accessible server and made available on request at no charge to the user.

Policies for Re-use, Distribution
Access to databases and associated software tools generated under the project will be available for educational, research and non-profit purposes. Such access will be provided using web-based applications, as appropriate.
Materials generated under the project will be disseminated in accordance with University/Participating institutional and NSF policies. Depending on such policies, materials may be transferred to others under the terms of a material transfer agreement.

Publication of data shall occur during the project, if appropriate, or at the end of the project, consistent with normal scientific practices. Research data which documents, supports and validates research findings will be made available after the main findings from the final research data set have been accepted for publication.

- Plans for Archiving and Preservation
  
  **Short Term:**
  The data product will be updated monthly reflecting updates to the record, revisions due to recalibration of standard gases, and identification and flagging of any errors. The date of the update will be included in the data file and will be part of the data file name. Versions of the data product that have been revised due to errors / updates (other than new data) will be retained in an archive system. A revision history document will describe the revisions made.

  Daily and monthly backups of the data files will be retained at the Keeling Group Lab (http://scrippsco2.ucsd.edu, accessed 05/2011), at the Scripps Institution of Oceanography Computer Center, and at the Woods Hole Oceanographic Institution’s Computer Center.

  **Long Term:**
  Our intent is that the long-term high quality final data product generated by this project will be available for use by the research and policy communities in perpetuity. The raw supporting data will be available in perpetuity as well, for use by researchers to confirm the quality of the Mauna Loa Record. The investigators have made arrangements for long-term stewardship and curation at the Carbon Dioxide Information and Analysis Center (CDIAC), Oak Ridge National Laboratory (see letter of support). The standardized metadata records for the Mauna Loa CO₂ data will be added to the metadata record database at CDIAC, so that interested users can discover the Mauna Loa CO₂ record along with other related Earth science data. CDIAC has a standardized data product citation [5] including DOI, that indicates the version of the Mauna Loa Data Product and how to obtain a copy of that product.