The integrated Rule-Oriented Data System (iRODS) 3.0 Micro-service Workbook

The integrated Rule-Oriented Data System (iRODS) 3.0 Micro-service Workbook

Copyright © 2011 by the DICE Foundation. All rights reserved.
Printed in the United States of America.

Published by the Data Intensive Cyberinfrastructure Foundation,
9700 Gilman Drive #175, La Jolla, California, 92093-5010 USA.

The integrated Rule-Oriented Data System (iRODS) organizes distributed data into sharable collections, while enforcing management policies. Each policy controls the execution of a procedure that processes the data at a remote storage location. Standard functions (called micro-services) are chained together into a workflow to compose these procedures. This book is a micro-service workbook, with descriptions of the input and output parameters and usage examples for each of the available micro-services. Together with the iRODS Primer, a community may use this book to assemble a data management infrastructure that reliably enforces their management and administrative policies and procedures.
## TABLE OF CONTENTS

PREFACE ................................................................................................................................. 1  
In This Workbook and Reference Manual .............................................................................. 1  
Platform Notes ....................................................................................................................... 1  
Typesetting ............................................................................................................................. 2  
Other Books ........................................................................................................................... 2  
Online Resources: the iRODS Wiki and Doxygen ............................................................... 3  
How to Contact Us .................................................................................................................. 4  
Acknowledgements ................................................................................................................ 4  
PART I A BRIEF INTRODUCTION TO iRODS ...................................................................... 5  
1.1 Micro-services Overview ............................................................................................... 5  
1.2 Data Grid Overview ....................................................................................................... 6  
1.3 iRODS Overview .......................................................................................................... 6  
1.4 Policy-based Data Management Overview ................................................................... 8  
1.5 The New Rule Engine .................................................................................................... 9  
1.6 The In-memory Rule Base .............................................................................................. 11  
1.7 Conclusion ....................................................................................................................... 13  
PART II A BRIEF INTRODUCTION TO iRODS RULE LANGUAGE ...................................... 15  
2.1 Comments and Directives ............................................................................................. 15  
2.2 Variables ......................................................................................................................... 16  
2.3 Data Types ..................................................................................................................... 16  
2.4 Control Structures ........................................................................................................ 21  
2.5 Functions ....................................................................................................................... 25  
2.6 Rules .............................................................................................................................. 25  
2.7 Types ............................................................................................................................. 26  
2.8 Micro-services .............................................................................................................. 29  
2.9 Rule Indexing ................................................................................................................ 30  
2.10 Backward Compatibility ............................................................................................... 31  
PART III iRODS MICRO-SERVICES CATEGORIES AND CONVE NITIONS ....................... 43  
3.1 Micro-services Overview ............................................................................................. 43  
3.2 Micro-services Categories ............................................................................................. 43  
3.3 The Micro-service Interface and msParam_t .................................................................. 44  
3.4 Micro-service Input/Output Parameters ....................................................................... 44  
3.5 How to Create a New Micro-Service ............................................................................ 45  
3.6 How to Interface a Micro-service as a Module ............................................................ 45  
3.7 How to Create a Module ............................................................................................... 45  
3.8 How to Use a Module's "info.txt" ................................................................................. 47  
3.9 How to Add a Micro-service to a Module .................................................................... 47  
3.10 How to Enable a Module ............................................................................................. 48  
3.11 How to Rebuild reAction.h .......................................................................................... 49  
3.12 How to Interface a System (Server) Micro-service ..................................................... 49  
3.13 Micro-service Naming Conventions ........................................................................... 50  
3.14 Micro-service Variable Naming Conventions ................................................................. 50  
3.15 Micro-service Constant Naming Conventions ............................................................... 50  
3.16 Micro-service Function Naming Conventions ............................................................... 50  
3.17 Micro-service File Naming Conventions ..................................................................... 51  
3.18 Examples of Writing Advanced Micro-Services ......................................................... 51  
3.19 Delaying the Execution of a Micro-Service ................................................................. 56  
3.20 Summary ..................................................................................................................... 57  
PART IV iRODS MICRO-SERVICES ..................................................................................... 59  
4.1 Core :: Collection :: msiAutoReplicateService ........................................................... 61  
4.2 Core :: Collection :: msiCollCreate ............................................................................ 62  
4.3 Core :: Collection :: msiCollRepl ................................................................................ 63  
4.4 Core :: Collection :: msiDataObjAutoMove ................................................................. 65  
4.5 Core :: Collection :: msiPhyBundleColl ...................................................................... 66
4.6 Core :: Collection :: msiRepColl ................................................................. 67
4.7 Core :: Collection :: msiRmColl ................................................................. 68
4.8 Core :: Collection :: msiTarFileCreate ....................................................... 69
4.9 Core :: Collection :: msiTarFileExtract ..................................................... 70
4.10 Core :: Database Object :: msiDboExec .................................................. 71
4.11 Core :: Database Object :: msiDbCommit ................................................ 72
4.12 Core :: Database Object :: msiDbRollback ............................................... 74
4.13 Core :: Data Object Low-level :: msiDataObjClose .................................... 75
4.14 Core :: Data Object Low-level :: msiDataObjCreate ................................... 76
4.15 Core :: Data Object Low-level :: msiDataObjLseek .................................... 77
4.16 Core :: Data Object Low-level :: msiDataObjOpen .................................... 78
4.17 Core :: Data Object Low-level :: msiDataObjRead .................................... 79
4.18 Core :: Data Object Low-level :: msiDataObjWrite .................................... 80
4.19 Core :: Data Object :: msiCheckAccess .................................................... 81
4.20 Core :: Data Object :: msiCheckOwner ...................................................... 82
4.21 Core :: Data Object :: msiCollRsync ......................................................... 83
4.22 Core :: Data Object :: msiDataObjChksum ............................................... 84
4.23 Core :: Data Object :: msiDataObjChksumWithOptions ............................... 85
4.24 Core :: Data Object :: msiDataObjCopy ................................................... 86
4.25 Core :: Data Object :: msiDataObjGet ..................................................... 87
4.26 Core :: Data Object :: msiDataObjGetWithOptions ................................... 88
4.27 Core :: Data Object :: msiDataObjPhymv ................................................. 89
4.28 Core :: Data Object :: msiDataObjPut ....................................................... 90
4.29 Core :: Data Object :: msiDataObjPutWithOptions .................................... 92
4.30 Core :: Data Object :: msiDataObjRename ............................................... 93
4.31 Core :: Data Object :: msiDataObjRepl ..................................................... 94
4.32 Core :: Data Object :: msiDataObjReplWithOptions .................................. 96
4.33 Core :: Data Object :: msiDataObjRsync ................................................... 97
4.34 Core :: Data Object :: msiDataObjTrim .................................................... 98
4.35 Core :: Data Object :: msiDataObjUnlink ................................................. 99
4.36 Core :: Data Object :: msiGetObjType ..................................................... 100
4.37 Core :: Data Object :: msiObjStat ........................................................... 101
4.38 Core :: Data Object :: msiPhyPathReg ..................................................... 102
4.39 Core :: Data Object :: msiSetReplComment ............................................. 103
4.40 Core :: Helper :: msiAddKeyValToMspStr ............................................... 104
4.41 Core :: Helper :: msiDoSomething .......................................................... 105
4.42 Core :: Helper :: msiGetSessionVarValue ............................................. 106
4.43 Core :: Helper :: msiGetStderrInExecCmdOut ....................................... 107
4.44 Core :: Helper :: msiGetStdoutInExecCmdOut ....................................... 108
4.45 Core :: Helper :: msiSplitPath ............................................................. 109
4.46 Core :: Helper :: msiWriteRodsLog ........................................................ 110
4.47 Core :: Helper :: msiExit ........................................................................ 111
4.48 Core :: Proxy Command :: msiExecCmd .................................................... 112
4.49 Core :: Rule Engine :: msiAdmAddAppRuleStruct ..................................... 113
4.50 Core :: Rule Engine :: msiAdmAddToTopOfCoreIRB .................................. 114
4.51 Core :: Rule Engine :: msiAdmAddToTopOfCoreRE .................................. 115
4.52 Core :: Rule Engine :: msiAdmChangeCoreIRB ......................................... 116
4.53 Core :: Rule Engine :: msiAdmChangeCoreRE .......................................... 117
4.54 Core :: Rule Engine :: msiAdmClearAppRuleStruct .................................. 118
4.55 Core :: Rule Engine :: msiAdmInsertRulesFromStructIntoDB ...................... 119
4.56 Core :: Rule Engine :: msiGetRulesFromDBIntoStruct ................................ 120
4.57 Core :: Rule Engine :: msiAdmReadRulesFromFileIntoStruct ...................... 121
4.58 Core :: Rule Engine :: msiAdmShowCoreRE ............................................ 122
4.59 Core :: Rule Engine :: msiAdmShowDVM ................................................. 123
4.60 Core :: Rule Engine :: msiAdmShowFNM ............................................... 124
4.61 Core :: Rule Engine :: msiAdmShowIRB .................................................. 125
4.62 Core :: Rule Engine :: msiAdmWriteRulesFromStructIntoFile ........................................126
4.63 Core :: Rule Engine :: msiAdmRetrieveRulesFromDBIntoStruct ...................................127
4.64 Core :: String Manipulation :: msiStrlen ......................................................128
4.65 Core :: String Manipulation :: msiStrchop ..................................................129
4.66 Core :: String Manipulation :: msiSubstr .....................................................130
4.67 Core :: Workflow :: assign ..............................................................................131
4.68 Core :: Workflow :: break .................................................................................132
4.69 Core :: Workflow :: cut .....................................................................................133
4.70 Core :: Workflow :: delay ..................................................................................134
4.71 Core :: Workflow :: errorCode .............................................................................136
4.72 Core :: Workflow :: fail .......................................................................................137
4.73 Core :: Workflow :: foreach .................................................................................138
4.74 Core :: Workflow :: for .........................................................................................139
4.75 Core :: Workflow :: if ............................................................................................140
4.76 Core :: Workflow :: applyAllRules .........................................................................141
4.77 Core :: Workflow :: msiGoodFailure ........................................................................142
4.78 Core :: Workflow :: msiSleep ................................................................................143
4.79 Core :: Workflow :: nop, null ...............................................................................144
4.80 Core :: Workflow :: print_hello ............................................................................145
4.81 Core :: Workflow :: remote ..................................................................................146
4.82 Core :: Workflow :: succeed ................................................................................147
4.83 Core :: Workflow :: while ......................................................................................148
4.84 Core :: Workflow :: writeLine ...............................................................................149
4.85 Core :: Workflow :: writeString ............................................................................150
4.86 Core :: XMessaging System :: msiCreateXmsgInp ..............................................151
4.87 Core :: XMessaging System :: msiRevXmsg .......................................................153
4.88 Core :: XMessaging System :: msiSendXmsg .......................................................155
4.89 Core :: XMessaging System :: msiXmsgCreateStream .........................................156
4.90 Core :: XMessaging System :: msiXmsgServerConnect .......................................157
4.91 Core :: XMessaging System :: msiXmsgServerDisConnect ....................................158
4.92 Core :: XMessaging System :: readXMsg .............................................................159
4.93 Core :: XMessaging System :: writeXMsg ...........................................................161
4.94 Core :: Framework Services System :: msiCheckHostAccessControl ...................162
4.95 Core :: Framework Services System :: msiDeleteDisallowed ................................163
4.96 Core :: Framework Services System :: msiDigestMonStat .....................................164
4.97 Core :: Framework Services System :: msiFlushMonStat .......................................166
4.98 Core :: Framework Services System :: msiListEnabledMS ....................................167
4.99 Core :: Framework Services System :: msiNoChkFilePathPerm .............................168
4.100 Core :: Framework Services System :: msiServerBackup ....................................169
4.101 Core :: Framework Services System :: msiSysMetaModify ..................................170
4.102 Core :: Framework Services System :: msiNoTrashCan .......................................171
4.103 Core :: Framework Services System :: msiOprDisallowed ....................................172
4.104 Core :: Framework Services System :: msiServerMonPerf .....................................173
4.105 Core :: Framework Services System :: msiSetBulkPutPostProcPolicy ..................174
4.106 Core :: Framework Services System :: msiSetDataObjAvoidResc ..........................175
4.107 Core :: Framework Services System :: msiSetDataObjPreferredResc .....................176
4.108 Core :: Framework Services System :: msiSetDataTypeFromExt ..........................177
4.109 Core :: Framework Services System :: msiSetDefaultResc ...................................178
4.110 Core :: Framework Services System :: msiSetGraftPathScheme ...........................179
4.111 Core :: Framework Services System :: msiSetMultiRep1PerResc ...........................180
4.112 Core :: Framework Services System :: msiSetNoDirectRescInp .............................181
4.113 Core :: Framework Services System :: msiSetNumThreads ....................................182
4.114 Core :: Framework Services System :: msiSetPublicUserOpr ...............................183
4.115 Core :: Framework Services System :: msiSetRandomScheme ............................184
4.116 Core :: Framework Services System :: msiSetRescQuotaPolicy ............................185
4.117 Core :: Framework Services System :: msiSetRescSortScheme ............................186
Core :: Other User :: writeBytesBuf
Core :: Other User :: writePosInt
Core :: Rule-oriented Database Access :: msiRdaCommit
Core :: Rule-oriented Database Access :: msiRdaNoResults
Core :: Rule-oriented Database Access :: msiRdaRollback
Core :: Rule-oriented Database Access :: msiRdaToDataObj
Core :: Rule-oriented Database Access :: msiRdaToStdout
Module :: ERA :: msiCollectionSpider
Module :: ERA :: msiCopyAVUMetadata
Module :: ERA :: msiCreateUserAccountsFromDataObj
Module :: ERA :: msiDeleteUsersFromDataObj
Module :: ERA :: msiExportRecursiveCollMeta
Module :: ERA :: msiFlagDataObjWithAVU
Module :: ERA :: msiFlagInfectedObjs
Module :: ERA :: msiGetAuditTrailInfoByActionID
Module :: ERA :: msiGetAuditTrailInfoByKey
Module :: ERA :: msiGetAuditTrailInfoByObjectID
Module :: ERA :: msiGetAuditTrailInfoByTimeStamp
Module :: ERA :: msiGetAuditTrailInfoByUserID
Module :: ERA :: msiGetCollectionACL
Module :: ERA :: msiGetCollectionContentsReport
Module :: ERA :: msiGetCollectionPSmeta
Module :: ERA :: msiGetCollectionSize
Module :: ERA :: msiGetObjectPath
Module :: ERA :: msiGetDataObjACL
Module :: ERA :: msiGetDataObjAIP
Module :: ERA :: msiGetDataObjAVUs
Module :: ERA :: msiGetDataObjPSmeta
Module :: ERA :: msiGetUserACL
Module :: ERA :: msiGetUserInfo
Module :: ERA :: msiGuessDataType
Module :: ERA :: msiIsColl
Module :: ERA :: msiIsData
Module :: ERA :: msiLoadACLFromDataObj
Module :: ERA :: msiLoadMetadataFromDataObj
Module :: ERA :: msiLoadUserModsFromDataObj
Module :: ERA :: msiMergeDataCopies
Module :: ERA :: msiRecursiveCollCopy
Module :: ERA :: msiSetDataType
Module :: ERA :: msiStripAVUs
Module :: ERA :: msiStructFileBundle
Module :: Guinot :: msiGetFormattedSystemTime
Module :: msoDrivers :: msiobjget_dbo
Module :: msoDrivers :: msiobjget_http
Module :: msoDrivers :: msiobjget_irods
Module :: msoDrivers :: msiobjget_slink
Module :: msoDrivers :: msiobjget_srb
Module :: msoDrivers :: msiobjget_test
Module :: msoDrivers :: msiobjget_z3950
Module :: msoDrivers :: msiobjput_dbo
Module :: msoDrivers :: msiobjput_http
Module :: msoDrivers :: msiobjput_irods
Module :: msoDrivers :: msiobjput_slink
Module :: msoDrivers :: msiobjput_srb
Module :: msoDrivers :: msiobjput_test
Module :: msoDrivers :: msiobjput_z3950

vii
<table>
<thead>
<tr>
<th>Page</th>
<th>Module :: HDF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.230</td>
<td>msiH5Dataset_read</td>
<td>hdf_read.h</td>
</tr>
<tr>
<td>4.231</td>
<td>msiH5Dataset_read_attribute</td>
<td>hdf_read_attribute.h</td>
</tr>
<tr>
<td>4.232</td>
<td>msiH5File_close</td>
<td>hdf_close.h</td>
</tr>
<tr>
<td>4.233</td>
<td>msiH5File_open</td>
<td>hdf_open.h</td>
</tr>
<tr>
<td>4.234</td>
<td>msiH5Group_read_attribute</td>
<td>hdf_group_read_attribute.h</td>
</tr>
<tr>
<td>4.235</td>
<td>msiImageConvert</td>
<td>image_convert.h</td>
</tr>
<tr>
<td>4.236</td>
<td>msiImageGetProperties</td>
<td>image_get_properties.h</td>
</tr>
<tr>
<td>4.237</td>
<td>rulegenerateBagIt.r</td>
<td>rule_generate_bag_it.r</td>
</tr>
<tr>
<td>4.238</td>
<td>ruleintegrityACL.r</td>
<td>rule_integrity_ACL.r</td>
</tr>
<tr>
<td>4.239</td>
<td>ruleintegrityAVU.r</td>
<td>rule_integrity_AVU.r</td>
</tr>
<tr>
<td>4.240</td>
<td>ruleintegrityExpiry.r</td>
<td>rule_integrity_expiry.r</td>
</tr>
<tr>
<td>4.241</td>
<td>ruleintegrityFileSize.r</td>
<td>rule_integrity_file_size.r</td>
</tr>
<tr>
<td>4.242</td>
<td>ruleintegrityFileOwner.r</td>
<td>rule_integrity_file_owner.r</td>
</tr>
<tr>
<td>4.243</td>
<td>listMS.r</td>
<td>list_MS.r</td>
</tr>
<tr>
<td>4.244</td>
<td>showCore.r</td>
<td>show_core.r</td>
</tr>
<tr>
<td>4.245</td>
<td>msiPropertiesClear</td>
<td>msi_properties_clear.h</td>
</tr>
<tr>
<td>4.246</td>
<td>msiPropertiesClone</td>
<td>msi_properties_clone.h</td>
</tr>
<tr>
<td>4.247</td>
<td>msiPropertiesExists</td>
<td>msi_properties_exists.h</td>
</tr>
<tr>
<td>4.248</td>
<td>msiPropertiesFromArray</td>
<td>msi_properties_from_array.h</td>
</tr>
<tr>
<td>4.249</td>
<td>msiPropertiesGet</td>
<td>msi_properties_get.h</td>
</tr>
<tr>
<td>4.250</td>
<td>msiPropertiesNew</td>
<td>msi_properties_new.h</td>
</tr>
<tr>
<td>4.251</td>
<td>msiPropertiesRemove</td>
<td>msi_properties_remove.h</td>
</tr>
<tr>
<td>4.252</td>
<td>msiPropertiesSet</td>
<td>msi_properties_set.h</td>
</tr>
<tr>
<td>4.253</td>
<td>msiPropertiesToString</td>
<td>msi_properties_to_string.h</td>
</tr>
<tr>
<td>4.254</td>
<td>msiFtpGet</td>
<td>msi_ftp_get.h</td>
</tr>
<tr>
<td>4.255</td>
<td>msiTwitterPost</td>
<td>msi_twitter_post.h</td>
</tr>
<tr>
<td>4.256</td>
<td>msiConvertCurrency</td>
<td>msi_convert_currency.h</td>
</tr>
<tr>
<td>4.257</td>
<td>msiGetQuote</td>
<td>msi_get_quote.h</td>
</tr>
<tr>
<td>4.258</td>
<td>msilp2location</td>
<td>msi_2location.h</td>
</tr>
<tr>
<td>4.259</td>
<td>msiObjByName</td>
<td>msi_obj_by_name.h</td>
</tr>
<tr>
<td>4.260</td>
<td>msiSdsSImgCutout_GetJpeg</td>
<td>msi_sds_img_cutout_get_jpeg.h</td>
</tr>
<tr>
<td>4.261</td>
<td>msiLoadMetadataFromXml</td>
<td>msi_load_metadata_from_xml.h</td>
</tr>
<tr>
<td>4.262</td>
<td>msiXmlDocSchemaValidate</td>
<td>msi_xml_doc_schema_validate.h</td>
</tr>
<tr>
<td>4.263</td>
<td>msiXmlDocSchemaValidate</td>
<td>msi_xml_doc_schema_validate.h</td>
</tr>
<tr>
<td>4.264</td>
<td>msi3950Submit</td>
<td>msi3950_submit.h</td>
</tr>
</tbody>
</table>

APPENDIX A: rodsPackTable.h DATA STRUCTURE TYPES ................................................................. 352
APPENDIX B: LIST OF msParam.c STRUCTURES .............................................................................. 353
APPENDIX C: LIST OF rodsPackInstruct.h PACKING INSTRUCTION DEFINITIONS ................................. 354
AUTHOR BIOGRAPHIES .................................................................................................................... 359
INDEX OF MICRO-SERVICES ........................................................................................................... 360
TABLE OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>iRODS' Peer-to-Peer Server Architecture</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>iRODS' Architecture Components</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>iRODS' Rule Engine Workflow</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>iRODS' In-memory Rule Base</td>
<td>12</td>
</tr>
</tbody>
</table>
PREFACE

Technologies for the management of distributed collections have evolved significantly over the last ten years. The original data grid software, the Storage Resource Broker (SRB), focused on the consistent management of properties of a collection across all operations performed upon collection contents. This meant that all policies were hard coded within the software framework to ensure that consistency guarantees could be met as files were moved between storage systems or modified. While the SRB has been highly successful software that has managed petabytes of data distributed around the world, each new application has required changes to the software framework.

The new generation of data grids, represented by the integrated Rule-Oriented Data System (iRODS), extracts policies from the software framework and manages them as computer actionable rules that are enforced by a distributed Rule Engine. This makes it possible to manage multiple sets of policies within a distributed collection, with local policies taking precedence. By extracting the policies from the data management framework, it became possible to build generic infrastructure that supports all phases of the data life cycle. Policies appropriate for controlling each stage can be implemented as the driving purpose behind the management of the collection evolves.

The development of the iRODS data grid has been strongly driven by requirements from multiple user communities. In particular, the application of iRODS in data grids, digital libraries, processing pipelines, and persistent archives has ensured that the data management goals of each community can be met. This has resulted in a software framework that is highly extensible, can be integrated with a wide variety of user interfaces, and can manage data stored in a wide variety of repositories. It is hoped that the use of policy-based data management systems will lead to self-consistent systems that are capable of verifying their internal properties and recovering from any detected problems. Such a system should be able to audit its trustworthiness, minimize the amount of administrative labor, and enforce community-specific policies.

In This Workbook and Reference Manual

The description of policy-based data management systems has been broken down into three separate topic areas.

- A general description of policy-based data management is available in the first book: iRODS Primer: integrated Rule-Oriented Data System. The iRODS Primer describes the rule language and collection properties and provides examples of policies for iRODS version 2.5 and earlier. While a list of basic functions (micro-services) is provided, no information is given on the input and output variables generated by each micro-service.

- This workbook and reference manual provides a detailed description of each basic function (micro-service) provided by the iRODS framework version 3.0. The basic functions can be chained together to create procedures that enforce a specific policy. Since there are a variety of desired policies, from automated replication to metadata extraction, from the creation of derived data products, to the validation of assessment criteria, there are many categories of micro-services. For each category, a description of the uses of the micro-services is provided. For each micro-service, a description of the input and output parameters is provided along with an example of the use of the micro-service within a rule.

- A third volume will describe the policies that are composed from micro-services that are in use in production data management environments. The reference book on iRODS policies will serve as a convenient guide for choosing policies to implement within a new data grid or digital library or preservation environment. The listed policies are intended to serve as a starter kit for creating community-specific policy sets.

In this iRODS micro-services workbook and reference manual, Part I provides an introduction to the concept of policy-based data management. Each policy controls a remote procedure that is executed at an iRODS server. Each procedure corresponds to a workflow that can query the metadata catalog, retrieve a
set of files, and then loop over the file set to implement a desired policy. A more detailed explanation is available in the iRODS Primer.

Part II describes the new rule language syntax. Examples are provided for converting the original rule language to the new syntax.

Part III lists the categories of micro-services and the conventions used to name parameters, session variables, and persistent state information. It also describes how to write a new micro-service.

Part IV describes the micro-services that are used to implement a procedure that can be run at a remote storage location. The micro-services are organized into categories related to core iRODS capabilities and modules that require integration with additional software systems. The core micro-services manage collections, manipulate database objects, manipulate data objects, provide support functions, manage the rule engine, manipulate strings, provide workflow functions, support the messaging system, support the iRODS framework, support metadata catalog queries (iCAT), send e-mail, process key-value pairs, support user functions, and support remote database access. Properties about users, collections, files, resources, and rules can be managed. General queries can be issued against the metadata catalog to extract sets of files for further processing. General framework support micro-services include setting the number of threads for parallel I/O streams, file staging, and file replication.

The module micro-services were designed for specific communities. These micro-services are organized into modules that may be added to the system at installation time. They include functions to support preservation procedures (ERA), parse time formats (Guinot), support access to remote information resources (msDrivers), support the Hierarchical Data Format (HDF), manipulate images (Image), create submission packages (integrity), manipulate property files (Properties), support URLs (URL), support web services (WebServices), manipulate XML (XML), and support Z39.50 access (Z3950).

The iRODS system uses in-memory structures to manage the exchange of information between micro-services. The in-memory structures can be serialized (through packing instructions) for transfer over a network. Appendix A lists the data structure types and the names of the packing instructions for serializing the data structures. This information is contained in the rodsPackTable.h file. Appendix B lists the msParam structures from "msParam.h" and the utility functions defined in "msParam.c" that manipulate the structures. Appendix C lists the packing instructions from "packInstruct.h". These three appendices can be used to determine the structure of all information exchanged between micro-services. Also included in this workbook are the Authors' Biographies.

Platform Notes

The micro-services that are described can be executed within an iRODS data grid version 3.0 or higher, as of the printing of this book. Note that the structures used to exchange information between micro-services continue to evolve. The structures used by the listed micro-services are specified in Part III and in the appendices. The structures required by micro-services in future releases of the iRODS software may be more sophisticated and require use of a newer version of iRODS.

Typesetting

The following conventions are used in this book.

1. iRODS Commands in examples and in text are in italics.

   Example 1:
   mycommandline% ils
   /zz/home/rods/t1:
   file1
   file2
Example 2: `icd "modules/MODNAME"`

2. **Micro-services in text and output from an i-command are in bold.**

Example 1: "FindObjectType" micro-service: **msiGetObjectType**.

Example 2:

```
mycommandline% ils /zz/home/rods/t1:
   file1
   file2
```

3. **Filenames and directory paths in text are in straight quotes."".**

Example 1: The "info.txt" file in a module's top-level directory describes the module.

Example 2: For instance, those in the "server/re/src" directory are part of the Rule Engine, whereas those in the "clients/icommands/src" directory are command-line tools.

4. **Code examples are indented.**

```
msiXsltApply(*xsltObjPath, *origObjPath, *BUF)
{
    msiDataObjCreate(*xmlObjPath,"null",*DEST_FD);
    msiDataObjWrite(*DEST_FD,*BUF,*Written);
    msiDataObjClose(*DEST_FD,*junk);
    msiLoadMetadataFromXml(*origObjPath, *xmlObjPath);
}
```

```
INPUT *origObjPath="orig.xml", *xmlObjPath="formatted.xml", *xsltObjPath="format_xml.xsl"
```

*Note that all input parameters for a rule are assumed to be entered on a single line. Code examples may appear to be on a new line, when in fact, the line has wrapped due to physical space limitations of this text.*

**Other Books**

The syntax used to express iRODS policies is a form of functional programming. More information about the syntax can be found at the iRODS wiki at http://irods.org.

Finally, while this book may serve as a micro-service workbook and reference manual for both new and experienced users of iRODS, all levels of users are encouraged to review the first iRODS book, the **iRODS Primer**. The **iRODS Primer** covers the following areas in greater depth than this book: *iRODS; the iRODS Architecture; Rule-Oriented Programming; the iRODS Rule System; Example Rules; Extending iRODS; iRODS Shell Commands; RuleGen Grammar;* and, **Exercises**.

You may locate this book via Morgan Claypool:


**Online Resources: the iRODS Wiki and Doxygen**

This micro-service reference book covers micro-services in more depth than the **iRODS Primer**. The reference material is current for iRODS version 3.0. The micro-service documentation is also available in
the iRODS code and online (http://irods.org/doxygen). While the content of this workbook is current with
the in-code documentation contained in the iRODS 3.0 release, additions and changes may be made to the
documentation at each iRODS release. Readers are encouraged to review the micro-service documentation
in the iRODS code or via the Doxygen output online for the latest versions.

Readers may also want to consider examining the iRODS wiki (http://irods.org) and the iRODS support
mailing list (http://groups.google.com/group/iROD-Chat/) for other iRODS-related information.

How to Contact Us

We have verified the information in this book to the best of our ability. If you find any errors, or have any
suggestions for future editions, you may reach us on the iRODS mailing list or via US mail at:

The Data Intensive Cyberinfrastructure Center
216 Manning Hall
University of North Carolina at Chapel Hill
Chapel Hill, North Carolina 27517
(919) 962-9548

Acknowledgements

This research was supported by:

NSF ITR 0427196, Constraint-Based Knowledge Systems for Grids, Digital Libraries, and Persistent

NARA supplement to NSF SCI 0438741, Cyberinfrastructure; From Vision to Reality— Developing

NARA supplement to NSF SCI 0438741, Cyberinfrastructure; From Vision to Reality— Research

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.
PART I A BRIEF INTRODUCTION TO iRODS

In this chapter:
- Micro-services Overview
- Data Grid Overview
- iRODS Overview
- Policy-based Data Management Overview
- The New Rule Engine
- The In-memory Rule Base
- Conclusion

Policy-based data management systems provide the essential capabilities required to automate administrative tasks, validate assessment criteria, and enforce institutional procedures. The integrated Rule-Oriented Data System (iRODS) represents the state-of-the-art in policy-based data management infrastructure in 2011. The iRODS data grid maps policies to machine actionable rules. Each rule controls the execution of a procedure that either applies an administrative task, or validates a property of a collection, or enforces a required property. The procedures are composed from basic operations that are chained together into workflows. Each basic operation is encapsulated within a C language function called a micro-service. The application of the micro-service generates state information that is managed by iRODS in a metadata catalog (iCAT). Validation of the properties of the collection is accomplished either through queries on the metadata catalog or through parsing of audit trails. By selecting the appropriate micro-services, any desired procedure can be implemented. By developing the appropriate rules, any desired policy can be enforced. Through use of the distributed data management capabilities of the iRODS data grid, the policies and procedures can be applied to data stored at multiple institutions across a wide variety of storage systems.

This book describes the set of micro-services that are provided with release 3.0 of the iRODS data grid. The iRODS software is distributed under an open source BSD license, and is available at http://irods.org. The listed micro-services are used in production applications of the iRODS software to support digital libraries, preservation environments, and data grids.

1.1 Micro-services Overview

The term "micro-service" refers to a C procedure that performs a simple task as part of a distributed workflow system. Each micro-service is small and well defined. Application programmers, systems administrators, and system programmers write and compile micro-services into server code within the integrated Rule-Oriented Data System (iRODS). This system is a community-driven, open source, middleware data grid that enables researchers, archivists and other managers to share, organize, preserve, and protect sets of digital files. The size of these sets may range from a few dozen files to hundreds of millions of files, and from a few megabytes in size to petabytes.

The iRODS framework is based upon 10 years of experience with the deployment and production of the Storage Resource Broker (SRB) data grid technology. The iRODS system combines ideas and technologies from the data grid, digital library, and archives domains to create the core concepts behind policy-based data management. Additional concepts and theories from within computer science include workflows, business rule systems, service-oriented architecture, active databases, transactional systems, constraint-management systems, logic programming, and program verification. The resulting system – iRODS – may be applied as: a digital library for publishing data; a persistent archive for preserving data; a system for large-scale data analysis; a system that gathers collections of real-time sensor data; and a data grid for sharing data.

Micro-services are the building blocks upon which procedures and management policies are implemented at the machine-level, regardless of the purpose of the particular iRODS instance. The micro-services can be reused by each new application through appropriate chaining into a procedure required by the new application domain policies. Thus, micro-services comprise the fundamental building blocks on which a policy-based data management system is built.
1.2 Data Grid Overview

A generic data grid is a system that virtualizes data collections. Data grids manage the properties of a data collection independently of the choice of storage system or database. One implication is that they can organize distributed data into a sharable collection. A generic data grid has three main characteristics. First, a data grid manages data distributed across multiple storage systems, such as tape archives, file systems, cloud storage, and institutional repositories. Second, it manages collection attributes such as system, descriptive metadata, and provenance information. Third, a data grid manages technology evolution, so that when software and hardware become obsolete, newer technology can be integrated with minimal effort.

If a data grid is to act as a digital library, a persistent archive, a workflow system for data analysis, or a real-time sensor system, the architecture must also have the ability to meet the diverse requirements of each of those systems. A digital library publishes data, and the system must support browsing and discovery. A persistent archive preserves data, and the system must be able to manage technology evolution. A workflow system provides the ability to analyze data, and the system must be able to integrate data management procedures with data analysis procedures. A real-time sensor system federates sensor data, and the system must be able to integrate data across sensor streams.

The architecture of a data grid that can meet all of the above requirements provides infrastructure independence, enforces management policies, manipulates structured information, is highly modular and extensible, provides scalability mechanisms, and enables community standards. In a production data grid, each community is able to implement different management policies that are specific to a collection or user group or storage resource or file type. Based on the management policies and preservation objectives, assertions about properties of the shared collection can be validated and shown to hold over time. The iRODS data grid provides all of these capabilities. As such, iRODS is capable of virtualizing the data life cycle through evolution of the policies and procedures that are used to manage the data collection.

1.3 iRODS Overview

The iRODS architecture belongs to the Adaptive Middleware Architecture (AMA) class. Middleware is generally designed as a black box that does not allow programmatic changes to the workflow, except for a few predetermined configuration options. Adaptive middleware provides a "glass box" such that users can examine the system processes, understand how they work, and adjust them to meet each users’ unique needs. The approach used to implement the AMA within iRODS is Rule-Oriented Programming (ROP). ROP provides the means for a user to customize data management functions by coding the processes that are being performed in iRODS as "rules". When an action is invoked by a particular task, these rules explicitly control the operations (micro-services) that are being performed. The execution of rules may be prioritized within the system so that one type of rule is executed prior to another type of rule. Similarly, users may modify the flow of tasks by adding, deleting, or modifying the micro-service(s) used within a rule; or by re-writing and recompiling, or deprecating the micro-service code itself.

The iRODS adaptive middleware architecture has three major features. First, the iRODS data grid architecture is based on a client-server model that manages the interactions among and between distributed compute and storage resources. Second, a metadata catalog, called the iCAT, maintains persistent state and data attributes generated by remote operations in a database of your choice. Third, a distributed Rule Engine controls the execution and enforcement of the Rules (Figure 1).
Figure 1 - iRODS' Peer-to-Peer Server Architecture

The iRODS Server software and the distributed Rule Engine are installed on each storage system. Thus, the elements of the iRODS system include: a Rule Engine and data grid server installed at each storage location; a Rule Base at each storage location that contains the available rules; a central iCAT Metadata Catalog; and multiple clients for accessing the data grid. The iCAT stores the persistent state information as attributes on users, attributes on collections, attributes on files, attributes on resources, and attributes on rules (Figure 2). iRODS uses over 200 attributes to manage information about each file, each collection, each storage resource, each user, and each rule. Examples of these stored attributes include the filename, owner, location, checksum, and data expiration date, among others.
In order to access an iRODS server, a user must be authenticated by the iCAT via information exchanged between the client, the iCAT catalog, and the iRODS server. The user may then proceed to invoke the desired procedures on that server. The Rule Engine at the server location applies additional constraints, controls the procedure execution, and passes the output of the operations back to the user’s client. The iRODS catalog stores any generated state information.

1.4 Policy-based Data Management Overview

iRODS automates the application of data management policies for pre- and post-processing, metadata extraction, loading, replication, distribution, retention, and disposition. The system also automates policies for services such as authentication, auditing, accounting, authorization, and administration. iRODS provides the ability to characterize the management policies required to enforce chain of custody, access restrictions, data placement, data presentation, integrity, and authenticity.

The simple core concepts driving the iRODS design are that every collection is created for a purpose, that every collection has management policies that enforce the collection purpose, and that every policy is implemented through a procedure (composed from micro-services). These fundamental concepts are implemented within iRODS technically by breaking them down into four distinguishing characteristics: workflow virtualization, management policy virtualization, service virtualization, and rule virtualization.

1. Workflow virtualization

The iRODS middleware enables the remote administration of collections of tightly controlled digital objects that are stored across heterogeneous storage locations. The data grid manages distributed objects and related metadata by providing infrastructure-independent procedures that can be run on any type of operating system. Effectively, the micro-services issue a standard API that is based on POSIX
I/O. The standard I/O calls are mapped to the storage access protocols required by each type of operating system. Thus, the same procedures are able to run on any computer. Micro-services are installed with each iRODS instance at each storage location. iRODS administrators and users "chain" the micro-services together to implement macro-level functions, called Actions. An iRODS administrator may control exactly which functions are executed at each storage location. These procedures may be executed remotely or locally. An administrator may implement a function multiple ways by creating multiple chains of micro-services for a particular Action. He or she may then set up the system to "choose" the best micro-service by setting up priorities and validation conditions that are evaluated at runtime.

2. **Management policy virtualization**

One way to think of priorities and validation conditions are that they are machine-level implementations of management procedures. These individual procedures are expressed in iRODS as management policies in the form of rules. The administrator controls the execution of a chain of one or more micro-services via these rules. For example, if the policy of a particular repository is to keep a log of all checksums associated with all files in directory "foo", an administrator would create a list of all checksums associated with the files by chaining together a series of micro-services in the form of a "rule" that would create a written log of the checksum values. Specifically, depending on the administrator's requirements, the rule might contain a micro-service that gets the checksum value for each file in directory "foo", a micro-service that finds the name of each associated file, a micro-service that finds the name of the associated directory (e.g., "collection"), and a micro-service that writes these values to a log file. The administrator is in control of the data grid, ensuring that even in a distributed environment, all files within a shared collection can be managed under the same policies.

3. **Service virtualization**

Rule-based data management is operationalized through micro-services. The concept behind the design of micro-services is that they have well-defined input-output properties, that policies are designed to support consistency verification, and that policies provide error recovery in the form of roll-back procedures. The compositional framework that micro-services provide is realized at runtime. One key capability of the iRODS design is that a user does not have to change any management policies during a micro-service upgrade. iRODS uses a logical name space for micro-services that provides the ability to organize and name them to ensure proper execution independent of any upgrades. The policies can be changed dynamically by modifying the rule base. It is possible to build an environment in which an original collection is managed by an original set of policies and procedures; a new collection can be defined that is controlled by new policies and procedures; and a policy can be written that manages the migration of files from the original collection to the new collection.

4. **Rule virtualization**

This is a logical name space for rules that provides the ability to version, name, and organize rules in sets that allows for the evolution of the rules themselves. Version 3.0 of iRODS supports the management of policies within the iCAT metadata catalog, supports the publication of policies to a remote storage location for inclusion within the local rule base, and supports versioning of policies.

1.5 **The New Rule Engine**

The rule engine is the interpreter of rules in the iRODS system. The rule engine can be invoked by any server-side procedure call, including the rule engine itself, using the rule engine application programming interface (API). The rule engine API supports execution of single actions and sequences of actions. An action usually consists of an action name and a list of arguments. Depending on the action name, executing the action may involve calling a micro-service or applying a rule.
Policy enforcement points are embedded in the iRODS data grid framework. When an action requested by a client traverses a policy enforcement point, the associated policy is invoked. The iRODS "core.re" file (local rule base) contains policies for each of the policy enforcement points. By convention, a standard set of policy names are used within the "core.re" file. Additional policies can be added to the rule base that can be invoked by explicit execution of a rule through the "irule" command.

An argument to an action may correspond to: an input parameter, whose value is passed from the calling routine to the rule engine; an output parameter, whose value is passed from the rule engine back to the calling routine; or a parameter that is both an input parameter and an output parameter. Figure 3, below, shows the workflow performed by the rule engine when a server-side procedure executes an action through the rule engine API.

![Figure 3 - iRODS' Rule Engine Workflow](image-url)
First, the rule engine tries to look up a rule whose rule name is the same as the action name. If a rule is found, it may cause multiple micro-services to be executed. If a micro-service succeeds, the micro-service should return a success status to the calling routine. If it fails, the micro-services can use the rule execution context provided by the rule engine to pass failure status and error messages back to the calling routine. If no micro-service is available, the rule engine selects all the rules whose rule names are the same as the action name that is passed in through the rule engine API.

The rules selected are prioritized based on how they are read into the rule base of the rule engine. The first rule in the list is checked for validation of its condition. If the condition fails, then the next rule is tried. If no more rules are available, then the action fails and a failure status (negative number) is returned to the calling routine. The rule engine performs optimizations to improve rule lookup efficiency when it loads rules into the rule base, but the optimization does not change the rule lookup semantics.

If the rule condition succeeds, then the micro-services in the rule are executed one after the other in the order they are given in the rule. For each action, the rule engine repeats the process described here recursively. If all of the micro-services in the rule succeed, then the rule execution is considered a success and a success status (usually 0) is returned to the calling routine. In such a case, the arguments that correspond to output parameters will hold any output values returned by the rule execution and the data structures holding the rule execution context will reflect any modifications that are made by the rule execution.

If one of the micro-services fails while executing the chain of actions, the rule engine starts a recovery procedure. For each micro-service that is executed, it applies the corresponding recovery micro-service defined in the rule. The recovery action for the failed action is performed first, followed by the recovery actions of all the previously successful actions in reverse order. The recovery actions can be defined so that they rollback any side-effects and restore the system to the initial state right before the rule execution.

1.6 The In-memory Rule Base

*Please note that "IRB" in the context of iRODS no longer means "iRODS Rule Base", it now means "In-memory Rule Base".*

The In-memory Rule Base is a structure managed in memory that contains the rules that are used in a session. The rules are automatically loaded into memory from a local "core.re" file on initiation of the session. In addition, `msiAdmAddAppRuleStruct` can be called to load additional rules for the session into the memory structure.

The persistent rules and rule versions are maintained in the iCAT catalog. To load rules into the iCAT, they must first be listed in a ".re" file. This file is loaded into memory using `msiAdmReadRulesFromFileIntoStruct`. The rules are then moved from memory into the iCAT catalog using `msiAdmInsertRulesFromStructIntoDB`. Rules that are loaded into the iCAT catalog cannot be removed. Instead, versions are created for each rule change to enable persistent governance of the data collection. The rule tables in the iCAT catalog comprise a persistent rule base.
The iRODS data grid manages distribution of rules from the persistent rule base into local "core.re" files. Each distributed rule engine reads a local "core.re" file to improve performance, and enable application of local rules. The process for creating a "core.re" file is the inverse of the process for loading rules into the iCAT catalog. `msiAdmRetrieveRulesFromDBIntoStruct` writes rules from the iCAT catalog into the In-memory Rule Base. `msiAdmWriteRulesFromStructIntoFile` can then be used to write a local rule (".re") file.

The In-memory Rule Base manages three types of rules:

1. Rules that are automatically loaded for the session from the local "core.re" file. These rules are indexed to improve rule engine performance.
2. Rules that are session-dependent that are loaded from an ancillary rule file using `msiAdmAddAppRuleStruct`.
3. Rules that are dynamically specified within an irule command.

A new "core.re" file is provided with release iRODS 3.0 for use by the new rule engine. If desired, the old rule engine and old rule language can be activated in iRODS 3.0. As an alternative, the new rule engine is able to parse the original legacy rule syntax, provided all string parameters are quoted. The iCAT stores all persistent rules and all versions of rules. Once a rule is checked into the iCAT, it can never be deleted.
1.7 Conclusion

The iRODS system of micro-service-based rules provides an extensible set of data management procedures targeted at programmers, users, and systems administrators who desire a variety of data management applications. iRODS is a generic software "middleware" infrastructure that can be adapted to the needs of individual data repositories, whether the need is for a collaborative data sharing environment, a digital library that publishes image, text and audio files, or a dark archive that must preserve and retain data for the indefinite long-term.
The iRODS Rule Language is a language provided by iRODS to define policies and actions in the system. The iRODS Rule Language is tightly integrated with other components of iRODS. Many frequently used policies and actions can be configured easily by writing simple rules, yet the language is flexible enough to allow complex policies or actions to be defined.

A typical rule written in the iRODS rule language looks like this:

```plaintext
acPostProcForPut {
    on($objPath like "*.txt") {
        msiDataObjCopy($objPath, "$objPath.copy");
    }
}
```

In this rule, the rule name "acPostProcForPut", meaning "post process for put", is an event hook defined in iRODS. iRODS automatically applies this rule when a file is "put" or uploaded into the system. The "on(...)" clause is a rule condition. The "{" block following the rule condition is a sequence of actions that is executed if the rule condition is true when the rule is applied. For another example, the customary hello world rule looks like this:

```plaintext
HelloWorld {
    writeLine("stdout", "Hello, world!");
}
```

In the 3.0 release of iRODS, a new rule engine is included that comes with an array of new features and improvements and takes care of some corner cases where the original rule engine was ambiguous, such as when parsing special characters in strings, by following the conventions of mainstream programming languages.

In the following sections, we go over some features of the new rule engine, with a focus on changes and improvements over the original rule engine.

### 2.1 Comments and Directives

The new rule engine parses characters between the "#" token and the end of the line as comments. Therefore, a comment does not have to occupy its own line. For example, the following comments are valid:

```plaintext
# comments
*A=1;
```

In the following sections, we go over some features of the new rule engine, with a focus on changes and improvements over the original rule engine.

### 2.1 Comments and Directives

The new rule engine parses characters between the "#" token and the end of the line as comments. Therefore, a comment does not have to occupy its own line. For example, the following comments are valid:

```plaintext
# comments
*A=1;
```
Although the parser is able to parse comments starting with "##", it is not recommended to begin comments with "##", as "##" is also used in the backward compatible modes as the actions connector (the original rule engine syntax used "##" as a delimiter). It is recommended to begin comments with a single "#".

Directives can be used to provide the rule engine with compile-time instructions. For example, the "@include" directive allows including a different rule file into the current rule file, similar to "#include" in C. For example, if we have a rule base file "definitions.re", then we can include it with the following directive:

@include "definitions"

2.2 Variables

Variables in the iRODS Rule Language are prefixed with "*" or "$". The scope of variables prefixed with "*" is a single top-level rule application. A variable may have any identifier as its name. The scope of variables prefixed with "$" is a session (which may contain several top-level rule applications). These variables are predefined by iRODS and cannot be created by a user.

Variables can be assigned values using "=":

*A=1;

They can also be used in expressions and actions:

*A+1
msi(*A);  # here msi is a micro-service that takes *A as an argument.

Rule parameters are also variables, and they are prefixed with "*". A rule parameter may be: an input parameter, whose value is passed from the calling rule to the rule being called; an output parameter, whose value is passed from the rule being called back to the calling rule; or a parameter that is both an input parameter and an output parameter:

div(*X, *Y, *Z) {
  *X = *Y / *Z;
}

"*Y" and "*Z" are input parameters and "*X" is an output parameter.

2.3 Data Types

2.3.1 Boolean

There are two boolean literals: "true" and "false". Boolean operators include "!" (not), "&" (and), "||" (or), and additionally in the original rule engine syntax "%" (or):

true & true
false & false
true || false
false || false
true % % false
false % % false
! true
2.3.2 Numeric

There are two primitive numeric data types: integer and double. The corresponding literals are integer literals and floating-point literals. An integral literal does not have a decimal, while a floating-point literal does:

```
1    # integer
1.0  # double
```

Arithmetic operators include, ordered by precedence from highest to lowest:

- (negation)
\^ (power)
* (multiplication) / (division) \% (modulo)
- (subtraction) + (addition)
> < >= <=
== !=

Note for C, C++, and Java programmers: the iRODS rule language does not implement integer division as found in C, C++, and Java; division between integers is the same as division between doubles.

Arithmetic functions include:

- exp
- log
- abs
- floor
- ceiling
- average
- max
- min

For example:

```
exp(10)
log(10)
abs(-10)
floor(1.2)
ceiling(1.2)
average(1,2,3)
max(1,2,3)
min(1,2,3)
```

In the iRODS Rule Language, an integer can be converted to a double. The reverse is not always true. A double can be converted to an integer only if the fractional part is zero. The rule engine, however, provides two functions that can be used to truncate the fractional part of a double: "floor" and "ceiling". Also, the numeric values 0 and 1 can be converted to booleans using the "bool" function. "bool" converts 1 to true and 0 to false.

2.3.3 String
One of the features in the new rule language that differs from the old rule engine is how it handles strings. The new rule engine requires, by default, that every string literal is quoted. The quotation marks can be either matching single quotes:

'This is a string.'

or double quotes:

"This is a string."

If a programmer needs to quote strings containing single (double) quotes using single (double) quotes, then the quotes in the strings should be escaped using a backslash "\" just as in C:

```c
writeLine("stdout", "\\\n");  # output is "\n"
```

Single quotes inside double quotes are viewed as regular characters, and vice versa. They can be either escaped or not escaped:

```c
writeLine("stdout", \\");  # output '  
writeLine("stdout", \\");  # output '
```

The rule engine also supports various other escaped characters:

```
n, r, \t, \v, \f, \", $, *
```

An asterisk should always be escaped if it is a regular character and is followed by letters.

The new rule engine supports the string concatenation operator "++":

```c
writeLine("stdout", "This "++" is "++" a string.");
# output This is a string.
```

wildcard matching operator "like":

```c
writeLine("stdout", "This is a string." like "This\*string.");
# output true
```

regular expression matching operator "like regex":

```c
writeLine("stdout", "This is a string." like regex "This.*string[.]");
# output true
```

substring function "substr":

```c
writeLine("stdout", substr("This is a string.", 0, 4));
# output This
```

length function "strlen":

```c
writeLine("stdout", strlen("This is a string."));
# output 17
```
and split function "split":

```plaintext
writeLine("stdout", split("This is a string.", ",");
# output [This, is, a, string.]
```

In a quoted string, an asterisk followed immediately by a variable name (without whitespace) makes an expansion of the variable:

"This is *x."

is equivalent to:

"This is "++str(*x)++"."

The "str" function converts a value of type boolean, integer, double, time, or string to string:

```plaintext
writeLine("stdout", str(123));
# output 123
```

A string can be converted to values of type boolean, integer, double, time, or string:

```plaintext
int("123")
double("123")
bool("true")
```

### 2.3.4 Rules for Quoting Action Arguments

A parameter to a micro-service is of type string if the expected type is MS_STR_T. When a micro-service expects a parameter of type string and the argument is a string constant, the argument has to be quoted. For example, `writeLine("stdout", "This is a string.").` When a micro-service expects a parameter of type string and the argument is not of type string, a type error may be thrown:

```plaintext
*x = 123;
strlen(*x);
```

This error can be fixed by either using the "str" function:

```plaintext
strlen(str(*x));
```

or by putting *x into quotes:

```plaintext
strlen("*x");
```

Action names and keywords are not arguments. Therefore, they do not have to be quoted.

### 2.3.5 Wildcard and Regular Expressions

The new rule engine supports both the wildcard matching operator "like" and a new regular expression matching operator "like regex" (it is an operator, not two separate keywords). Just as the old rule engine does, the new rule engine supports the "**" wildcard:

"abcd" like "ab**
In case of ambiguity with variable expansion, the "\*" must be escaped:

"abcd" like "a\*d"

because:

"a*d"

is otherwise interpreted as:

"a"++str(*d)+"

When a wildcard is not expressive enough, the regular expression matching operator can be used:

"abcd" like regex "a.c."

A regular expression matches the whole string. It follows the syntax of the POSIX API.

2.3.6 Quoting Code

Sometimes when passing code or regular expressions into an action, escaping every special character in the string can be very tedious:

"writeLine("stdout", \*A)"

or:

*A like regex "a\*c\\\\\\n" # matches the regular expression a*c\\\\

In this case, matching sets of two back ticks (```) can be used instead of the regular double quotes. The rule engine does not look any further for things to expand within strings between two ```s. With ```, the examples above can be written as:

``writeLine("stdout", *A)`

and:

``A like regex `a*\c\\\\[]`''

2.3.7 Lists

The new rule engine provides built-in support for lists. A list can be created using the "list" micro-service:

list("This","is","a","list")

All elements of a list should have the same type. Elements of a list can be retrieved using the "elem" micro-service. The index starts from 0:

elem(list("This","is","a","list"),1)    # evaluates to "is"

If the index is out of range it fails with an error code.
The "setelem" micro-service takes three parameters, a list, an index, and a value, and returns a new list that is identical to the list given by the first parameter except that the element at the index given by the second parameter is replaced by the value given by the third parameter:

```plaintext
setelem(list("This","is","a","list"),1,"isn't") # evaluates to list("This","isn't","a","list")
```

If the index is out of range it fails with an error code.

The "size" micro-service takes one parameter, a list, and returns the size of the list:

```plaintext
size(list("This","is","a","list")) # evaluates to 4
```

The "hd", or head, micro-service returns the first element of a list and the "tl", or tail, micro-service returns the remainder of the list. If the list is empty, then both fail with an error code.

```plaintext
hd(list("This","is","a","list")) # evaluates to "This"
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
A rule consists of a sequence of actions. Each action can apply a rule or execute a micro-service. For example, if there is a nullary micro-service named "msi" and a binary rule named "rule", then the following actions apply the rule and execute the micro-service:

```
rule(*A, *B);
msi;
```

There is a set of familiar workflow micro-services, such as "assign", "if", and "foreach", that have special syntax support. For example:

```
if(*A==0) {*B=true;} else {*B=false;}
```

The iRODS rule engine has a unique concept of recovery action. Every action in a rule definition may have a recovery action. The recovery action is executed when the action preceding it fails. This allows iRODS rules to rollback some side effects and restore most of the system state to a known previous point. The new rule engine supports a more general notion of an action recovery block. An action recovery block has the form:

```
{ 
   A_1 ::: R_1 
   A_2 ::: R_2 
   ... 
   A_n ::: R_n 
}
```

The basic semantics are that if $A_1$ fails then $R_1$, $R_{n}$, ..., and $R_1$ will be executed. The programmer can use this mechanism to restore the system state to the point before this action recovery block began to be executed.

The new rule engine makes the distinction between expressions and actions. An expression does not have a recovery action. Examples of expressions include the rule condition, and the conditional expressions in the "if", "while", and "for" actions. An action always has a recovery action. If a recovery action is not specified for an action, the rule engine will use "nop", or no operation, as the default recovery action:

```
{ 
   msi; 
}
```

is equivalent to:

```
{ 
   msi; ::: nop; 
}
```

There is no intrinsic difference between an action and an expression. An expression becomes an action when it occurs at an action position in an action recovery block. An action recovery block, in turn, is an expression.

The principle is that an expression should only be used not as an action if it is side-effect free. This property is not checked in the current version of the rule engine (v3.0). The programmer has to make sure that it holds for the rule base being executed.
2.4.2 "if"

The new rule engine has a few useful extensions to the "if" keyword that makes programming in the rule language more convenient.

In addition to the traditional way of using "if" in the rule language, the new rule engine supports a new way of using "if". The traditional way will be referred to as the "logical if", where "if" is used as an action which either succeeds, or fails with an error code. The new way will be referred to as the "functional if", where it may return a value of any type if it succeeds. The two different usages have slightly different syntaxes.

The "logical if" has the same syntax as before:

```
if <expr> then { <actions> } else { <actions> }
```

The "functional if" has the following syntax:

```
if <expr> then <expr> else <expr>
```

For example, the following is written as an older "logical if":

```
if (*A==1) then { true; } else { false; }
```

And now the same written as a newer "functional if":

```
if *A==1 then true else false
```

To make the syntax of "logical if" more concise, the new rule engine allows the following abbreviations:

- `if (...) { ... } else { ... }
- if (...) then { ... } else if (...) then {...} else {...}

Multiple abbreviations can be combined, for example:

```
if (*X==1) { *A = "Mon"; }
   else if (*X==2) { *A = "Tue"; }
   else if (*X==3) { *A = "Wed"; }   …
```

2.4.3 "while"

The syntax for "while" is:

```
while(<expr>) { <actions> }
```

The semantics are similar to C. The actions are performed iteratively until the expression becomes false:

```
*X = 0;
*S = 0;
*N =5;
while(*X < *N) {
    *S = *S + *X;
    *X = *X + 1;
}
```
2.4.4 "for"

The syntax for "for" is:

\[
\text{for}(\text{<expr>};\text{<expr>};\text{<expr>}) \{ \text{<actions>} \}
\]

Initially, the first expression is evaluated. The actions are performed iteratively, until the second expression becomes false. The third expression is evaluated after each iteration.

For example,

\[
*\text{S} = 0;
*\text{N} = 5;
\text{for}(*\text{X} = 0;*\text{X} < *\text{N}; *\text{X} = *\text{X} + 1) \{
\text{<actions>} \\
*\text{S} = *\text{S} + *\text{X};
\}
\]

2.4.5 "foreach"

The syntax for "foreach" is:

\[
\text{foreach}(\text{<variable>}) \{ \text{<actions>} \}
\]

The variable has to have a collection type such as the general query result type or a list type, something that can be iterated through by the foreach. The foreach action goes through all elements stored in the variable and performs the actions, during which the variable is bound to the current element.

The new rule engine allows defining a different variable name for the iterator variable in the foreach action:

\[
\text{foreach}(\text{<variable>} \text{ in } \text{<expr>}) \{ \text{<actions>} \}
\]

For example:

\[
\text{foreach}(*\text{E} \text{ in } *\text{C}) \{ \\
\text{writeLine("stdout", *\text{E});} \\
\}
\]

This is equivalent to the earlier syntax:

\[
\text{foreach}(*\text{C}) \{ \\
\text{writeLine("stdout", *\text{C});} \\
\}
\]

This new feature allows the collection to be a complex expression:

\[
\text{foreach}(*\text{E} \text{ in list("This", "is", "a", "list"))} \{ \\
\text{writeLine("stdout", *\text{E});} \\
\}
\]

This is equivalent to the earlier syntax:

\[
*\text{C} = \text{list("This", "is", "a", "list");} \\
\text{foreach}(*\text{C}) \{
\}
2.4.6 "let"

The syntax for the let expression is:

```
let <assignment> in <expr>
```

For example:

```
quad(*n) = let *t = *n * *n in *t * *t
```

The variable on the left hand side of the assignment in the let expression is a let-bound variable. A let-bound variable should not be reassigned inside the let expression.

2.5 Functions

The new rule engine allows defining functions. Functions can be thought of as micro-services written in the rule language. The syntax of a function definition is:

```
<name>(<param>, …, <param>) = <expr>
```

For example:

```
square(*n) = *n * *n
```

Function names should be unique within the rule engine (no function-function or function-rule name conflicts).

Functions can be defined in a mutually exclusive manner:

```
odd(*n) = if *n==0 then false else even(*n-1)
even(*n) = if *n==0 then true else odd(*n-1)
```

To use a function, call it as if it was a micro-service.

2.6 Rules

The syntax of a rule with a nontrivial rule condition is as follows:

```
<name>(<param>, …, <param>) { 
on(<expr>) { <actions> } 
}
```

If the rule condition is trivial or unnecessary, the rule can be written in the simpler form:

```
<name>(<param>, …, <param>) { <actions> }
```

Multiple rules with the same rule name and parameters list can be combined in a more concise syntax where each set of actions is enumerated for each set of conditions:
\[\text{<name>(<param>, \ldots, <param>) \{\n  \text{on(<expr>) \{ <actions> \}}\n  \ldots\n  \text{on(<expr>) \{ <actions> \}}\n}\]

### 2.6.1 Rule Name

In the new rule engine, rule names have to be valid identifiers. Identifiers start with letters followed by letters or digits:

ThisIsAValidRuleName

There should not be whitespace in the rule name:

This Is Not A Valid Rule Name

### 2.6.2 Rule Condition

In the new rule engine, rule conditions should be expressions of type boolean. The rule is executed only when the rule condition evaluates to true. This means that there are three failure conditions:

1. The rule condition evaluates to false.
2. An action in the rule condition fails which causes the evaluation of the entire rule condition to fail.
3. The rule condition evaluates to a value whose type is not boolean.

For example, if we want to run a rule when the micro-service "msi" succeeds, we can write the rule as:

```
  testrule { 
    on (msi >= 0) { ... } 
  }
```

Conversely, if we want to run a rule when the micro-service fails, we need to write the rule as:

```
  testrule { 
    on (errorcode(msi) < 0) { ... } 
  }
```

The following rule condition always fails by failure condition 3 listed above because msi returns an integer value, not a boolean value:

```
  on(msi) { ... }
```

### 2.7 Types

#### 2.7.1 Introduction

Types are useful for capturing errors before rules are executed, but a restrictive type system may also rule out meaningful expressions. As the rule language is a highly dynamic language, the main goals of introducing a type system are are twofold:

1) To enable the discovery of some errors statically without ruling out most valid rules written for the old rule engine, and
2) To help remove some repetitive type checking and conversion code in micro-services by viewing types as contracts of what kinds of values are passed between the rule engine and micro-services.

The type system is designed so that the rule language is dynamically typed when no type information is given, while providing certain static guarantees when some type information is given. The key is combining static typing with dynamic typing, so that we only need to check the statically typed part of a program statically and leave the rest of the program to dynamic typing.

The new rule engine distinguishes between two groups of micro-services. System-provided micro-services such as string operators are called internal micro-services. The rest are called external micro-services. Most internal micro-services are statically typed. They come with type information which the type checker can make use of to check for errors statically. By default, external micro-services are dynamically typed, but they can be assigned a static type by a programmer using type declaration.

A type declaration specifies types for parameters of micro-services and rules and their return values. The primitive types include: boolean, integer, double, time, string, and iRODS types. From the primitive types, complex types such as list types, tuple types, and algebraic data types can be built. A type called dynamic is included for dynamically typed values.

Typing constraints are used in the new rule engine to encode typing requirements that need to be checked at compile time or at runtime. The type constraints are solved against a type coercion relation, a model of whether one type can be coerced to another type and how their values should be converted. For example, in the new rule engine integers can be implicitly coerced to doubles, but not the other way around.

### 2.7.2 Variable Typing

As in C, all variables in the rule language have a fixed type that can not be updated through an assignment. For example, the following does not work:

```java
testTyping1 {}  
  *A = 1;   
  *A = "str";
}
```

Once a variable *A is assigned a value X the type of the variable is given by a typing constraint:

- type of X can be coerced to type of *A

For example:

```java
testTyping2 {}  
  *A = 1;   # integer can be coerced to type of *A  
  *A = 2.0; # double can be coerced to type of *A
}
```

Solving the typing constraints, we have:

- type of *A must be double

For another example, the following generates a type error:

```java
testTyping3 {}  
  *A = 1; # integer can be coerced to type of *A
```
if(*A == "str") { # type error occurs here
}
}

If the value of a variable is dynamically typed, then a coercion is inserted. The following example works, with a runtime coercion:

testTyping4 {
    msi(*A);
    if(*A == "str") { # insert coercion from type of *A to string
        ...
    }
}

2.7.3 Types by Examples

In this subsection, we look at a few simple examples of how the new rule engine works with types. In the new rule engine, binary arithmetic operators such as addition and subtraction are given the type: 

    forall X in \{integer double\}, f X * f X > X 

This type indicates that the operator takes in two parameters of the same type and returns a value of the same type as its parameters. The parameter type is bound by \{integer double\}, which means that the operator applies to only integers or doubles. The "f" indicates that if any type can be coerced to these types, it can also be accepted with a runtime coercion inserted.

Examples:

(a) When both parameter types are double, the return type is also double:

    1.0 + 1.0 # returns double

(b) When one of the parameter types is integer and the other is double, the return type is double, because integer can be coerced to double, but not conversely:

    1 + 1.0 # returns double

(c) When both parameter types are integer, the return type is integer, which can also be coerced to double:

    1 + 1 # returns integer

(d) If one of the parameter types is dynamic, and the other is double, the return type is double, with a runtime constraint:

    *A + 1.0 # returns double

The type checker generates a constraint that the type of *A can be coerced to double.

(e) If both parameter types are dynamic, the return type can be either integer or double:

    *A + *B # unclear return type without more context
The type checker generates a constraint that both type of \( *A \) and type of \( *B \) can be coerced to either integer or double.

Some typing constraints can be solved within a certain context. For example, we put (e) into the following context:

\[
* B = 1.0; \\
* B = *A + *B; \quad \# \text{returns double}
\]

We can eliminate the possibility that \( *B \) is an integer, thereby narrowing the type of the return value to double.

Some typing constraints can be proved unsolvable:

\[
* B = *A + *B; \\
* B == ""
\]

By the second action we know that \( *B \) has to have type string. In this case the rule engine reports a type error.

If some typing constraints can not be solved statically, they are left to be solved at runtime.

2.7.4 Type Declaration

In the new rule engine, you can declare the type of a rule or a micro-service. The syntax for type declaration is:

\[
<\text{name}> : <\text{type}>
\]

A typical \( <\text{type}> \) looks like:

\[
P_1, P_2, \ldots, P_n \rightarrow R
\]

\( P_1, P_2, \ldots, \) and \( P_n \) are the parameter types and \( R \) is the return type.

If the type of an action is declared, then the rule engine will do more static type checking. For example, although this does not generate a static type error:

\[
\text{concat}(a, b) = a ++ b \\
\text{add}(a, b) = \text{concat}(a, b)
\]

"add(0, 1)" would generate a dynamic type error. This can be solved (generate static type errors instead of dynamic type errors) by declaring the types of the functions:

\[
\text{concat} : \text{string} \times \text{string} \rightarrow \text{string} \\
\text{concat}(a, b) = a ++ b \\
\text{add} : \text{integer} \times \text{integer} \rightarrow \text{integer} \\
\text{add}(a, b) = \text{concat}(a, b)
\]

2.8 Micro-services
2.8.1 Automatic Evaluation of Arguments

The new rule engine automatically evaluates expressions within arguments of actions, which is useful when a program needs to pass in the result of an expression as an argument to an action. For example, in the old rule engine, if we want to pass the result of an expression "1+2" as an argument to micro-service "msi", then we need to either write something like this:

```c
*A=1+2;
msi(*A);
```

Or, we have to pass "1+2" in as a string to "msi" and write code in the micro-service which parses and evaluates the expression. With the new rule engine, the programmer can write:

```c
msi(1+2);
```

and the rule engine will evaluate the expression "1+2" and pass the result of 3 into the micro-service.

2.8.2 The Return Value of User Defined Micro-services

Both the old rule engine and the new rule engine view the return value of user defined micro-services as an integer "errorcode". If the return value of a micro-service is less than zero, both rule engines interpret it as a failure, rather than as an integer value; and if the return value is greater than zero, both rule engines interpret it as an integer. Therefore, the following expression:

```c
msi >= 0
```

either evaluates to true or fails (and never evaluates to false), because when "msi" returns a negative integer, the rule engine interprets the value as a failure and the comparison is never evaluated. In some applications, there is a need for capturing all possible return values as regular integers. The "errorcode" micro-service provided by the new rule engine can be used to achieve this. In the previous example, we can modify the code as follows:

```c
errorcode(msi) >= 0
```

This expression does not fail on negative return values from msi and then allows the greater than or equal to comparison to be evaluated as expected.

2.9 Rule Indexing

To improve the performance of rule execution, the new rule engine provides a two level indexing scheme on applicable rules. The first level of indexing is based on the rule names. The second level of indexing is based on the rule conditions. The rule condition indexing can be demonstrated by the following example:

```c
testRule(*A) {
  on (*A == "a") { ... }
  on (*A == "b") { ... }
}
```

In this example, we have two rules with the same rule name, but different rule conditions. The first level of indexing does not improve the performance in rule applications like:

```c
testRule("a")
```
However, the second level indexing does improve performance. The second level indexing works on rules with similar rule conditions. In particular, the rule conditions have to be of the form:

\[
\text{<expr>} == \text{<string>}
\]

The syntactical requirement for the rule indexing to work are: all rules have to have the same number of parameters, but they may have different parameter names; the expression has to be the same for all rules modulo variable renaming; and the strings have to be different for different rules. The rule engine indexes the rules by the string. When the rule is called, the rule engine evaluates the expression once and looks up the rule using the second level indexing.

Rule indexing also works on subsets of rules that satisfy the syntactical requirement:

```plaintext
testRule(*A) {
  on (msi(*A)) { ... }
  on (*A == "a") { ... }
  on (*A == "b") { ... }
}
```

Rule indexing works on the second and third rule, but not the first. When the rule is called, the rule engine tries the first rule first, if the first rule fails, it tries to look up an applicable rule using the second level indexing.

### 2.10 Backward Compatibility

#### 2.10.1 Backward Compatibility Mode

The new rule engine has backward compatibility modes that allow it to run rules written for the old rule engine in the "##" syntax and a less strict grammar with little or no changes necessary. The backward compatibility mode can be set to "true", "false", or "auto" using the `@backwardCompatible` directive:

```plaintext
@backwardCompatible "true"
acPostProcForPut|objPath like *.txt|writeLine(serverLog, text: $objPath)|nop
@backwardCompatible "false"
acPostProcForPut {
  on($objPath like "*.html") {
    writeLine("serverLog", "html: $objPath");
  }
}
```

As shown in the example, the backward compatibility mode status can be mixed within one code base. Note the unquoted strings in the first half of the example and the required quoted strings in the second half.

By default, the backward compatibility mode is set to "auto". In the "auto" mode, the rule engine tries to detect whether the code is written in the "##" syntax or the newer syntax and automatically apply backward compatibility to the "##" syntax.

Note that the example above can equivalently be written as:

```plaintext
@backwardCompatible "auto"
acPostProcForPut|objPath like *.txt|writeLine(serverLog, text: $objPath)|nop
acPostProcForPut {
  on($objPath like "*.html") {
```
Additionally, the "auto" line could be omitted and the default compatibility setting would generate the same runtime effect.

2.10.2 Limitations of Backward Compatible Modes

2.10.2.1 Variables in Like Expressions

In the old rule engine, the following code matches $objPath with the pattern "*txt":

```java
acPostProcForPut|$objPath like *txt|nop|nop
```

In the new rule engine, even with the backward compatibility mode turned on, it matches $objPath with the contents of the *txt variable, as the "*" character is followed by a letter. (Note: If the string is "*.txt", with the quotation marks, then there is no ambiguity.) To match with the pattern "*txt" while using the new rule engine, change the rule to include a backslash:

```java
acPostProcForPut|$objPath like "\*txt"|nop|nop
```

The "\" character escapes the "*" character following it and turns it from a variable prefix into a regular character, in this case, one that matches any other characters when evaluated by the rule engine.

2.10.2.2 Micro-services in Rule Conditions

The old rule engine allows the following code:

```java
rule|msiDoSomething|nop|nop
```

The rule is executed only if the micro-service "msiDoSomething" succeeds. A micro-service is considered successful if it returns an integer greater than or equal to 0 and failed if it returns an integer less than 0. The new rule engine requires the rule condition to be a boolean expression which returns either true or false. We could add the following implicit conversion rules:

- `integer >= 0` -> `true`
- `integer < 0` -> `false`

However, this would be inconsistent with the conventions of C and C++, where we are used to:

- `integer != 0` -> `true`
- `integer == 0` -> `false`

This would lead to deterministic, but confusing behavior. Therefore, we did not include those two implicit conversion rules. The example, however, can be written in the new rule engine as follows:

```java
rule {
    on(msiDoSomething >= 0) {
    }
}
```

To test for the failure case, you can use the `errorcode` function:
rule {
    on(errorcode(msiDoSomething) < 0) {
    }
}

2.10.2.4 Expressions in irule Input Parameters

The old rule engine allows the input parameters to be either an unquoted string or an expression:

testrule|writeLine(stdout, *A *D)|nop
*A=unquoted string%*D=0 + 1
ruleExecOut

While the old rule engine returns:

unquoted string 1

The new rule engine returns:

unquoted string 0 + 1

In the new rule engine, if you use the "##" syntax, then all values of input parameters are strings, quoted or unquoted. In the newer syntax, it can be written as:

testrule {
    writeLine("stdout", "*A *D");
}  
input *A="unquoted string", *D=0 + 1
output ruleExecOut

Note that rules written in the "##" syntax have to be saved in a "*.ir" file and rules written in the newer notation have to be saved in a "*.r" file.

2.10.3 Converting from the Old Rule Engine Syntax to the New Rule Engine Syntax

The old rule engine provides two syntaxes for writing rules. The first syntax (the "##" syntax), as found in the "core.irb" file, looks like this:

acPostProcForPut|$objPath like *.txt|msiDataObjCopy($objPath, "$objPath.copy")|nop

It has the restriction that every rule must be written in one line for fast processing. The second syntax (originally known as the rulegen syntax) is a more readable, more approachable format and supports multi-line rules:

acPostProcForPut {
    on($objPath like *.txt) {
        msiDataObjCopy($objPath, $objPath.copy);
    } 
}

However, rules written in the rulegen syntax have to be preprocessed using the rulegen tool into the "##" syntax before the old rule engine can process them. Therefore, they cannot be directly included in the "core.irb" file.
In order to bring the rule language closer to the conventions of mainstream programming languages and provide a simple and complete solution for eliminating ambiguity, such as special characters in strings, the new rule engine requires a slightly more rigorous syntax than the rulegen syntax. For example, all strings must be quoted in the new rule engine.

To make it easier to migrate rules written in the "##" syntax, the new rule engine provides backward compatibility modes. Backward compatibility modes tweak the rule engine parser and type checker so that they simulate the old rule engine when parsing the "##" syntax. In backward compatibility mode, strings do not have to be quoted. Backward compatibility modes work only with the "##" syntax. Backward compatibility mode can be turned on, off, or to automatic within a code file using the @backwardCompatible directive. Automatic mode allows users to have both types of rules in a single code base. The backward compatibility mode and its limitations are explained in the previous section.

The old rule engine itself is also included in the 3.0 release in case full backward compatibility is needed. The rule engine used by iRODS can be easily switched to the old one by making a small change in the server Makefile and rebuilding the server. Obviously, if this is done, no new rule engine syntax or typing will be available.

This section explains how to convert a rule written in the "##" syntax to the new rule engine syntax. First, we look at a rule written in the "##" syntax. In the example, the first four lines should be concatenated into a single line for execution. The entire rule consists of three lines: the code to be executed, the inputs, and then the outputs:

```
My Test Rule(*arg)|msi(*arg) && *arg like *txt|delayExec(<A></A>,
copyDataObj(*objPath)##moveDataObj(*objPath), nop##nop)##remoteExec(localhost, null,
writeLine(stdout, *D), nop)##assign(*A, "a, b, c")##assign(*B, *A string)##forEachExec(*A,
writeLine(serverLog, *A), nop)##assign(*A, "a, b, c")##assign(*B, *A string)##forEachExec(*A,

*obj=test.txt%*D=string
```

Next, we will convert this rule into the new rule engine syntax. There are two main steps. The first main step is to convert the rule from "##" syntax to the rulegen syntax, the same as in iRODS 2.5 and before. The second main step converts differences between the rulegen syntax and the new rule engine syntax.

### 2.10.3.1 From "##" to Rulegen

1. Insert newlines. (This step is optional.)

```
My Test Rule(*arg)|msi(*arg) && *arg like *txt|
delayExec(<A></A>,
copyDataObj(*objPath)##moveDataObj(*objPath), nop##nop)##remoteExec(localhost, null,
writeLine(stdout, *D), nop)##assign(*A, "a, b, c")##assign(*B, *A string)##forEachExec(*A,
writeLine(serverLog, *A), nop)##assign(*A, "a, b, c")##assign(*B, *A string)##forEachExec(*A,

*obj=test.txt%*D=string
```

ruleExecOut
2. Convert "|"s to the rulegen syntax.

My Test Rule(*arg) {
    on(msi(*arg) && *arg like *txt) {
        delayExec(<A></A>,
        copyDataObj(*objPath),
        moveDataObj(*objPath),
        nop;nop);#
        remoteExec(localhost, null,
        writeLine(stdout, *D),
        nop);#
        assign(*A, "a, b, c");#
        assign(*B, *A string);#
        forEachExec(*A,
        writeLine(serverLog, *A),
        nop)
    }
}  
*obj=test.txt%*D=string
ruleExecOut

3. Convert "##"s to ","s.

My Test Rule(*arg) {
    on(msi(*arg) && *arg like *txt) {
        delay(<A></A>,
        copyDataObj(*objPath);
        moveDataObj(*objPath),
        nop;nop);
        remoteExec(localhost, null,
        writeLine(stdout, *D),
        nop);
        assign(*A, "a, b, c");
        assign(*B, *A string);
        forEachExec(*A,
        writeLine(serverLog, *A),
        nop)
    }
}  
*obj=test.txt%*D=string
ruleExecOut

4. Convert delayExec, remoteExec, assign, forEachExec, etc. to rulegen syntax.

In this step, you can add or remove ";"s to make it follow standard C conventions.

My Test Rule(*arg) {  
on(msi(*arg) && *arg like *txt) {  
    delay(<A></A>) {  
        copyDataObj(*objPath):::nop;
        moveDataObj(*objPath):::nop;
    }  
    remoteExec(localhost, null) {  
        writeLine(stdout, *D):::nop;
    }  
}
5. Convert input and output to rulegen syntax.

```c
My Test Rule(*arg) {
  on(msi(*arg) && *arg like *txt) {
    delay(<A><A>) {
      copyDataObj(*objPath):::nop;
      moveDataObj(*objPath):::nop;
    }
    remoteExec(localhost, null) {
      writeLine(stdout, *D):::nop;
    }
  }
  *A = "a, b, c";
  *B = *A string;
  foreach(*A) {
    writeLine(serverLog, *A):::nop;
  }
}
}
```

input *obj=test.txt, *D=string
output ruleExecOut

6. Delete superfluous nops. (This step is optional.)

```c
My Test Rule(*arg) {
  on(msi(*arg) && *arg like *txt) {
    delay(<A><A>) {
      copyDataObj(*objPath);
      moveDataObj(*objPath);
    }
    remoteExec(localhost, null) {
      writeLine(stdout, *D);
    }
  }
  *A = "a, b, c";
  *B = *A string;
  foreach(*A) {
    writeLine(serverLog, *A);
  }
}
}
```

input *obj=test.txt, *D=string
output ruleExecOut

The rule has now been converted into the old rule engine rulegen syntax. The next series of steps bring the rule into the new rule engine syntax.
7. Quote strings.

The general guideline is simple, if some text is not quoted, then it is parsed as code. If you want to pass some argument to a micro-service and do not want the rule engine to interpret it, then quote the argument to turn it into a string. For example:

```plaintext
writeLine(serverLog, *A);
```

Here, the word "serverLog" is interpreted by the rule engine as a micro-service or rule because it is not quoted. The rule engine will try to execute that micro-service or rule and will pass the return value in as the argument. To pass the string "serverLog" as the argument, it has to be quoted:

```plaintext
writeLine("serverLog", *A);
```

Similarly, the arguments to remote and delay are also strings. So, they need to be quoted.

On the right hand side of an assign statement, we also need to quote the string. If the right hand side is an expression we do not quote it. For example:

```plaintext
*A = "0 + 1"
```

assigns the string “0 + 1” to *A, and

```plaintext
*A = 0 + 1
```

assigns the integer 1 to *A.

On the right hand side of the like expression, we need to quote the pattern because it is also a string.

The input parameters are also strings, and we need to quote them.

The rule now looks like this:

```plaintext
My Test Rule(*arg) {
    on(msi(*arg) && *arg like "*txt") {
        delay("<A></A>") {
            copyDataObj(*objPath);
            moveDataObj(*objPath);
        }
        remoteExec("localhost", "null") {
            writeLine("stdout", *D);
        }
        *A = "a, b, c";
        *B = "*A string";
        foreach(*A) {
            writeLine("serverLog", *A);
        }
    }
    *arg="test.txt", *D="string"
}
```
8. Escape special characters in strings.

Special character such as "*" may be interpreted by the rule engine even within other quoted strings.

For example:

*arg like "*txt"

Here, "*txt" is considered a variable which is expanded into the string. However, what we intended to do is to use "*txt" as a pattern. Therefore, the "*" should not be interpreted by the rule engine. To prevent interpretation we convert it to "\*txt".

```
My Test Rule(*arg) {
  on(msi(*arg) && *arg like "\*txt") {
    delay("<A></A>") {
      copyDataObj(*objPath);
      moveDataObj(*objPath);
    }
    remoteExec("localhost", "null") {
      writeLine("stdout", *D);
    }
    *A = "a, b, c";
    *B = "*A string";
    foreach(*A) {
      writeLine("serverLog", *A);
    }
  }
}
input *arg="test.txt", *D="string"
output ruleExecOut
```

9. Delete white spaces in rule names.

Rule names must be valid identifiers. Now, the rule looks like this:

```
MyTestRule(*arg) {
  on(msi(*arg) && *arg like "\*txt") {
    delay("<A></A>") {
      copyDataObj(*objPath);
      moveDataObj(*objPath);
    }
    remoteExec("localhost", "null") {
      writeLine("stdout", *D);
    }
    *A = "a, b, c";
    *B = "*A string";
    foreach(*A) {
      writeLine("serverLog", *A);
    }
  }
}
input *arg="test.txt", *D="string"
output ruleExecOut
```
10. Use "split" to split the string if it is used as a collection in foreach.

In the new rule engine, "foreach" requires the parameter to be a list or iRODS type "GenQueryOut_PI". The case where it is a comma separated string can be simulated using "split":

```
MyTestRule(*arg) {
    on(msi(*arg) && *arg like "\*txt") {
        delay("<A></A>") {
            copyDataObj(*objPath);
            moveDataObj(*objPath);
        }
        remoteExec("localhost", "null") {
            writeLine("stdout", *D);
        }
        *A = split("a, b, c", ", ");
        *B = "*A string";
        foreach(*A) {
            writeLine("serverLog", *A);
        }
    }
}
```

input *arg="test.txt", *D="string"
output ruleExecOut

11. Convert micro-service calls in rule conditions.

The micro-service call

```
msi(*arg)
```

in the rule condition returns an integer. If the integer is greater than or equal to 0 then the micro-service is considered successful; otherwise it is considered failed. Therefore, when we write in the old syntax

```
msi(*arg) && *arg like "\*txt"
```

we are not trying to compute the "logical and" with the return code of "msi(*arg)", but rather with whether "msi(*arg)" succeeds.

In the new rule engine, we make this explicit by writing:

```
msi(*arg) >= 0 && *arg like "\*txt"
```

The operators ">=\" and "like\" have higher priority than "&&\" and are evaluated first.

Now the rule looks like this:

```
MyTestRule(*arg) {
    on(msi(*arg) >= 0 && *arg like "\*txt") {
        delay("<A></A>") {
            copyDataObj(*objPath);
            moveDataObj(*objPath);
        }
    }
}
```
12. Remove arguments from the main rule.

This step only applies to the first rule in an input file to the irule command.

In the new rule engine, all input variables and output variables are global variables. And the first rule is called as the "main" rule in the irule command. The "main" rule should not have any parameters:

```plaintext
MyTestRule {
  on(msi(*arg) >= 0 & & *arg like "\(^*\)") {
    delay("<A> </A>" ) {
      copyDataObj(*objPath);
      moveDataObj(*objPath);
    }
    remoteExec("localhost", "null") {
      writeLine("stdout", *D);
    }
    *A = split("a, b, c", ", ");
    *B = "*A string" ;
    foreach(*A) {
      writeLine("serverLog", *A);
    }
  }
}
```

input *arg="test.txt", *D="string"
output ruleExecOut

Now we have a syntactically valid rule for the new rule engine. The rule has to be saved in a ".r" file as the irule command determines whether an input is in the new rule engine syntax or the "##" syntax by a file's extension.

Other things to consider:

- The "like" expression only supports one type of wildcard ":*". If your rule uses other kinds of wildcards, it can be converted to the regular expression
- If you still get a type error, you can try to explicitly convert the value to the correct type using one of the following functions: "int", "double", "bool", or "str". For example, if you have an error with:

  ```plaintext
  *A = 1;
  msi(*A);
  ```
and msi expects a string argument, you can convert it with:

```
msi(str(*A));
```

- The iRODS Wiki may have more updates and examples on how to migrate rules written for the old rule engine to the new rule engine.
PART II

IRODS MICRO-SERVICES CATEGORIES AND CONVENTIONS

In this chapter:
- Micro-services Overview
- Micro-services Categories
- Micro-services Input/Output Arguments
- Micro-services Naming Conventions
- Examples of Writing Advanced Micro-services
- Summary

3.1 Micro-services Overview

Micro-services are small, well-defined C procedures (functions) developed by systems programmers and applications programmers to perform a certain task. This task may be very complicated or quite small. If a task is large, it may be best to be divided into smaller tasks as multiple micro-services. However, if two sub-tasks are usually used together, they may be best combined into a single micro-service. There is tension between making a large task into a single micro-service that does not allow an end user or administrator to choose which part of a task to run, and creating micro-services that are so fine-grained that the implementation of the task becomes cumbersome. As with all programming, using normal coding practices and good design principles in deciding the granularity of a micro-service are recommended. The micro-services are compiled into the iRODS server code for installation at each storage location. The data grid administrator has control over the operations that will be performed within the data grid.

3.2 Micro-services Categories

Micro-services are divided into the following four categories:

1. **Core Micro-services** - These micro-services are functions for Rule Engine control, workflow creation, and low-level and higher-level data object manipulation. Low-level data operations include opening a file, closing a file, reading data, and writing data. High-level data operations include replication, checksumming, registration, and the staging of files.

2. **Framework Services** - These are functions for rule-oriented remote database access, message passing with the high-performance Xmessaging system, sending e-mail, manipulating Keyword–Value attribute pairs, supporting user-defined services, and supporting system level services. An example system level service is the vacuuming of a postgres database.

3. **iCAT Services** - These are functions for manipulating system metadata, and for interacting with the iCAT.

4. **Module Micro-services** - These are sets of functions developed for specific communities, for example, the ERA (Electronic Records Archives) Program at the National Archives and Records Administration (NARA), eXtensible Markup Language (XML) manipulation, Hierarchical Data Format (HDF) manipulation, image property manipulation, Web service interaction, the French National Library, etc.

If larger macro functionality is required, then end users and administrators may chain together a series of micro-services. This macro-level functionality provides full control over any actions performed. An end user or administrator may also chain together different micro-services to provide different ways to perform the same or similar action. The system itself "chooses" the best micro-service chain to be executed using priorities and validation conditions at run-time.

Some examples of micro-services are: `msiCreateUser, msiDeleteUser, msiTarFileExtract`, `msiTarFileCreate, msiCollCreate`, and `msiRenameCollection`. A complete list of micro-services
available at publication time is available in the index of micro-services. The most recent list of micro-services may be found online at https://irods.org/doxygen/.

### 3.3 The Micro-service Interface and `msParam_t`

Any normal C procedure can be a micro-service, but there is a standard interfacing template that needs to be used. The C procedure you want to use as a micro-service can have any number of arguments, type or structure. The iRODS Rule Engine interacts with a micro-service through the `msParam_t` structure. This is a published parametric structure that is standardized within iRODS. A micro-service interface (msi) is used to convert from `msParam_t` to the argument types required by the underlying C code within the micro-service. The msi routine maps `msParam_t` to the call arguments and converts back any output parameters to the `msParam_t` structure; it is glue code. Thus, a C procedure called "createUser" will have an interface routine called `msicreateUser`; the Rule Engine invokes `msicreateUser`, and this invokes the "createUser" function. Only registered micro-services will be invoked by the Rule Engine; thus, each msi procedure must be registered by adding the name of the msi procedure call to a C structure table, `reAction.table[]`, which is located in iRODS/server/re/include/reAction.table.

`msParam_t` provides the following uniform type definitions for use by the Rule Engine to handle the distributed operation of micro-services:

```c
typedef struct MsParam {
    char *label;
    char *type;    /* This is the name of the packing instruction in *rodsPackTable.h*/
    void *inOutStruct;
    bytesBuf_t *inpOutBuf;
} msParam_t;
```

In the example above, "label" is the name of the argument in the call; "type" is the C structure type supported by iRODS; "inOutStruct" is a pointer to the value of the input structure being passed (it can be "NULL"); and "inpOutBuf" is used to specify any binary buffers that need to be passed as part of the argument. Each type has a packing instruction that defines how the structure can be serialized for transmission over the network.

### 3.4 Micro-service Input/Output Parameters

For input/output parameters, a user or administrator may pass a variable to a micro-service through explicit arguments, exactly as in the case of C function or procedure calls. The input parameters may take two forms:

- **Literal**: If an argument does not begin with a special character (#, $ or *), it is treated as a character string input if it is quoted. For example, in the micro-service `msiSortDataObj("random")`, the character string "random" will be passed in as input. Literals can only be used as input parameters and not as output parameters. Literals can be of type string, integer, and double.

- **Variable**: If an argument begins with the * character, it is treated as a variable argument. Variable arguments can be used both as input and output parameters. The output parameter from one micro-service can be explicitly specified as the input parameter of another micro-service. This powerful capability allows very complex workflow-like rules to be constructed.

For example, in the following workflow chain:

```c
myTestRule {
    # Input parameters:
    # targetObj - iRODS target file that metadata will be attached to
    # xmlObj - iRODS path to XML file that metadata is drawn from
```
msiLoadMetadataFromXml uses as the input variable the referenced "*xmlObj" file, extracts the AVU metadata from the "xmlObj" file, and registers the metadata onto the file name represented by "*targetObj" in the iCAT catalog. For the full example and the XML file, please see Module :: XML :: msiLoadMetadataFromXml in section 4.209.

In all of the rule examples, the INPUT variables are listed on a single line. Also the OUTPUT variables are listed on a single line. Thus in the example above, the INPUT line contains both the *xmlObj and *targetObj variables on a single line and is written:

```plaintext
INPUT *xmlObj="/tempZone/home/rods/XML/sample-processed.xml", *targetObj="/tempZone/home/rods/XML/example.xml"
```

For information on how to pass arguments to a rule or action, please see Section 6.1 "Micro-service Input/Output Arguments" in the iRODS Primer.

### 3.5 How to Create a New Micro-Service

Micro-services can be added as either system micro-services in the server area, or as module micro-services that are compiled only as necessary for a given application. We discuss both options below.

### 3.6 How to Interface a Micro-service as a Module

This requires a two-step process:

- Create a module, which is done once for each module.
- Add a micro-service to the module, which is done for every new micro-service that is added to the module.

### 3.7 How to Create a Module

A "module" is a bundle of optional software components for iRODS. Typically, a module provides specialized micro-services. Modules also may provide new rules, library functions, commands, and even application servers. Once you have developed the software to perform a new iRODS function, you can add your software as a new iRODS module with the following steps:

1. Create a directory named for the module, in this case MODNAME:
mkdir iRODS/modules/MODNAME

2. Move into that directory:

   cd iRODS/modules/MODNAME

3. Create one or more subdirectories for the components being added to iRODS:

   mkdir microservices
   mkdir rules
   mkdir lib
   mkdir clients
   mkdir servers

For the rest of these instructions, we will assume you are adding micro-services, but similar instructions apply for the other types of additions.

4. Create source, include, and object subdirectories:

   mkdir microservices/src
   mkdir microservices/include
   mkdir microservices/obj

5. Add your source and include files to the "src" and "include" directories.

6. Create a makefile by copying one from an existing module, such as "properties".

   cp ..../properties/Makefile .

7. Edit the makefile to list your source files and add any special compile flags or libraries you may need. The makefile must respond to a set of standardized targets:

   all build everything
   microservices build new micro-services
   client build new clients
   server build new servers
   rules build new rules
   clean remove built objects, etc.
   client_cflags compile flags for building clients
   client_ldflags link flags for building clients
   server_cflags compile flags for building servers
   server_ldflags link flags for building servers

The micro-services, client, and server targets should compile your code. The client and server targets should link your custom clients and servers. If your module does not have one or more of these, the target should exist but do nothing.

The client and server flag targets should echo to stdout the compiler or linker flags needed on other clients and servers that use the module. The "cflags" echos should list -I include paths and specific include files. The "ldflags" echos should list -L link paths, -l library names, and specific library or object files.

8. Create an "info.txt" file by copying one from an existing module:

   cp ..../properties/info.txt .
9. Edit the "info.txt" file to include information about your module. The file must contain:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>the name of the module</td>
</tr>
<tr>
<td>Brief:</td>
<td>a short description of the module</td>
</tr>
<tr>
<td>Description:</td>
<td>a longer description of the module</td>
</tr>
<tr>
<td>Dependencies:</td>
<td>a list of modules this module needs</td>
</tr>
<tr>
<td>Enabled:</td>
<td>whether the module is enabled by default</td>
</tr>
</tbody>
</table>

Each of these must be on a single (possibly long) line. For dependencies, list module names separated by whitespace. Module names must match exactly the directory name of other modules, and the names are case sensitive. The "enabled" flag is either "yes" or "no".

3.8 How to Use a Module's "info.txt"

The "info.txt" file in a module's top-level directory describes the module. It is intended for use by the iRODS makefile and configuration scripts.

The file is a list of keyword-value pairs, one per line. For instance:

```
Name: Sample
Brief: A sample micro-service module
Description: This is a sample module description
Dependencies:
Enabled: yes
Creator: University of North Carolina at Chapel Hill
Created: Sept 2011
License: BSD
```

**Name:** The name of the module. The name should match the module directory name.

**Brief:** A brief half-line description of the module. The iRODS configure script uses this value when printing help information about available modules.

**Description:** A longer description of the module. While the value must be on a single line, it can be several sentences long.

**Dependencies:** A list of module names upon which the module depends. Names should be space-separated and must match module directory names. The iRODS configure script uses this to insure that all modules that must be enabled together are enabled together.

**Enabled:** The value "yes" or "no" indicates whether the module should be enabled by default. The iRODS configure script uses this to set defaults on configuring iRODS.

**Creator:** The name of the principal individual or organization responsible for creating the module.

**Created:** The approximate creation date of the module.

**License:** The license covering the module's source code. Additional information may be in the source files or in a module's documentation. This value is only used as a general indicator.

A Doxygen application processes the source code, extracting text fields for documentation. The default Doxygen script processes comments that are inserted about each micro-service. Check existing modules for examples of the preferred format if you are planning to contribute your module back to the community.

3.9 How to Add a Micro-service to a Module
All micro-service functions are discovered by the iRODS server by reading a master "action" table compiled into the server. The action table is split into three files:

1. server/re/include/reAction.header
2. server/re/include/reAction.table
3. server/re/include/reAction.footer

The iRODS makefiles assemble these files and those provided by modules to build the file "server/re/include/reAction.h". This file contains:

- reAction.header: The header for each micro-service module.
- reAction.table: The table entries for each micro-service module.
- reAction.footer: Text following the table entries.

There are two ways to add a micro-service.

1. For system micro-services:
   - Edit “reAction.header” to add function prototypes.
   - Edit “reAction.table” to add table initializations.

2. For module micro-services:
   - Create “MODNAME/microservices/include/microservices.header”. Edit this to add function prototypes.
   - Create “MODNAME/microservices/include/microservices.table”. Edit this to add table initializations.

Function prototypes declare the C micro-service function. While these can be added to the above files directly, authors are encouraged to use a separate include file and just add an #include for that file. For instance, here is a typical "microservices.header" file for a module:

```c
// Sample module microservices
#include "sampleMS.h"
```

The "reAction.table" and each module's "microservices.table" file contains a C array initialization listing all available micro-services. Each line in the initialization looks like this:

```c
// Sample module microservices
{ "msisample", 2, (funcPtr) msiSample }
```

There are three values, in order:

1. The service name is the user-visible name of the micro-service. It is a string using only letters, numbers, and underscores. It should be descriptive and need not match the micro-service function name.

2. The argument count is the number of msParam_t arguments for the function. It does not include the ruleExecInfo_t argument.

3. The function name is a pointer to the C function for the micro-service.

3.10 How to Enable a Module

Remember that you must also turn on a module to use it. There are three ways you can enable a module.
1. Before running `irodssetup` or `irodsupgrade`, edit the module's "info.txt" file and set "Enabled: yes". This makes the iRODS configuration default to enabling the module. If your system is already compiled, you just need to rerun iRODS/scripts/configure after editing your module's "info.txt" file. Then rerun `make` from the iRODS directory to recompile. This is the preferred method for enabling a module and you will see the enabled modules listed.

2. Edit the module by running the iRODS configure script:

   ```
   iRODS/scripts/configure --enable-MODNAME
   ```

   This enables the module for your installation. The script sets the MODULES variable in "config/config.mk". Then rerun `make` from the iRODS directory to recompile.

3. Edit "config/config.mk" directly (discouraged) and add your module's name to the MODULES variable. Then rerun `make` from the iRODS directory to recompile.

### 3.11 How to Rebuild `reAction.h`

The server's "reAction.h" action table is rebuilt automatically by the iRODS root makefile if there are any changes in "server/re/include" or "modules/*/microservices/include". To force "reAction.h" to be rebuilt, delete the existing file and run the "reaction" makefile target:

```
rm server/re/include/reAction.h
make reaction
```

### 3.12 How to Interface a System (Server) Micro-service

1. Create the micro-service function as needed.

   ```
   int myPetProc(char *in1, int in2, char *out1, int *out2)
   {
      ... my favorite code ...
   }
   ```

2. Create the micro-service interface (msi) glue procedure.

   ```
   int msiMyPetProc(msParam_t *mPin1, msParam_t *mPin2,
                    msParam_t *mPout1, msParam_t *mPout2,
                    RuleExecInfo_t *rei)
   {
      char *in1, out1;
      int i, in2, out2;

      RE_TEST_MACRO (" Calling myPetProc")
      /* the above line is needed for loopback testing using the irule -i option */

      in1  = (char *) mPin1->inOutStruct;
      in2  = (int)    mPin2->inOutStruct;
      out1 = (char *) mPout1->inOutStruct;
      out2 = (int)    mPout2->inOutStruct;

      i  = myPetProc(in1, in2, out1, &out2);
      mPout2->inOutStruct = (int) out2;

      return(i);
   }
   ```
3. Define the msi call in the file "server/re/include/reAction.table" by adding the function signature in the area where all function signatures are defined:

```c
int msiMyPetProc(msParam_t *mPin1, msParam_t *mPin2, msParam_t *mPout1, msParam_t *mPout2, RuleExecInfo_t *rei);
```

4. Register the micro-service by making an entry in the file "server/re/include/reAction.table". The first item in the entry is the external name of the micro-service, the second is the number of user-defined arguments for the msi procedure call (excluding the "RuleExecInfo_t *rei"), and the third argument is the name of the msi procedure. Note that the names are the same in the following example for the first and third values in the entry. We recommend this format for clarity purposes:

```c
{"msiMyPetProc", 4, (funcPtr) msiMyPetProc}
```

5. If there are any "include" files that are needed, they can be added to "server/re/include/reAction.header".

6. Define the called procedure in an appropriate include file (for the present "reFuncDefs.h" file would be a reasonable place for this, since this will require no change in any makefile) by adding the signature.

```c
int myPetProc(char *in1, int in2, char *out1, int *out2);
```

The micro-service is now ready for compilation and use.

### 3.13 Micro-service Naming Conventions

When users or systems administrators add files and functions, we recommend using standard naming conventions for ease of maintenance. Following a standard naming convention is useful for maintaining the programs and functions our users create. While we do not force these conventions on volunteer developers, we recommend their usage for maintaining good programming practice.

### 3.14 Micro-service Variable Naming Conventions

We recommend that variable names use multiple descriptive words.

**Example:** myRodsArgs

Variable names use camel-case to distinguish words, with the first letter of each word component capitalized.

**Example:** genQueryInp

### 3.15 Micro-service Constant Naming Conventions

We recommend using one of the two following conventions:

1. Constant string names use multiple descriptive words and start with an uppercase letter.

   **Example:** Msg_Header_PI

2. Constant string names use uppercase letters separated with an underscore.

   **Example:** NAME_LEN

### 3.16 Micro-service Function Naming Conventions
All C functions in iRODS occupy the same namespace. To avoid function name collisions, we recommend that:

- Function names use multiple descriptive words.
  Example: getMsParamByLabel
- Function names use camel-case to distinguish words.
  Example: printMsParam
- Micro-service function names start with "msi".
  Example: msiDataObjGet
- Micro-service helper function names start with "mh".
  Example: mht
- Server function names start with "rs".
  Example: rsGetXmsgTicket
- Client function names start with "re".
  Example: rcCollCreate

3.17 Micro-service File Naming Conventions

The purpose of a file may be inferred by the location of the file in the iRODS directory tree. For instance, those in the "server/re/src" directory are part of the Rule Engine, whereas those in the "clients/icommands/src" directory are command-line tools. Beyond this, we recommend that:

- File names use multiple descriptive words.
  Example: rodsServer.c contains the iRODS Server main program.
- File names reflect the names of functions in the file.
  Example: msParam.c contains utility functions that work with the msParam structure.
- File names use camel-case to distinguish words.
  Example: irodsReServer.c
- No two files in the same directory may have names that differ only by case. Case-insensitive names cause problems with Windows and legacy Mac (OS 9 and earlier) file systems.

3.18 Examples of Writing Advanced Micro-Services

The challenge of writing a new micro-service can be broken down into three tasks:

1. Coding modules to be micro-service compliant.
2. Testing micro-services from the command line.
3. Using the micro-services as a workflow.
For a simple example problem we will prototype how to extract metadata Attribute-Value pairs from one file based on a template defined in a second file, and associate the metadata to a third file.

We will base the design on use of a template that defines the patterns that are parsed from the metadata file. The template file identifies the Keyword-Value pairs that may be present in a document. A <PRETAG> defines the pattern that precedes the desired metadata. A <POSTTAG> defines the pattern that follows the desired metadata. The name that will be used for the metadata attribute is listed between the <PRETAG> and <POSTTAG> pairs.

For example:

```xml
<PRETAG>Date: </PRETAG>SentDate<POSTTAG> </POSTTAG>
<PRETAG>From: </PRETAG>Sender<POSTTAG> </POSTTAG>
```

Whatever is found between "Date:" and " " provides the "value" for the keyword "SentDate".

The metadata files contains the actual metadata that needs to be ingested, and is typically a text file such as an e-mail message:

```
Date: Thu, 01 Feb 2008, 22:33:35 +000
From: adil Hasan <a.hasan@rl.ac.uk>
To: Michael Wan <mwan@sds.edu>
```

We want to design a micro-service that will extract the "Date" and name of the person who sent the email.

Instead of writing a micro-service for file-metadata ingestion only, we will design a more widely useful set of micro-services that can be applied to any iRODS object (data, collection, resource, user, token or metadata). The implementation will involve the following four steps.

1. Define the signature of the micro-service.
2. Register the micro-service as an invokable method by the iRODS Rule Engine.
3. Create the micro-service. This may need other function calls (new and old).
4. Describe the micro-service.

We will create two micro-services:

1. "FindObjectType" micro-service: `msiGetObjType`.

The implementation requires creating a "Signature Definition".

All micro-devices have only TWO types of parameters or attributes.

1. Params 1…(n-1) are of the type `msParam_t`.
2. Param n is of the type `ruleExecInfo_t`.

`msParam_t` is defined as:

```c
typedef struct MsParam {
    char *label;
    char *type;
    void *InOutStruct;
    bytesBuf_t *inpOutBuf;
} msParam_t;
```
**ruleExecInfo_t** is the "white board" used for passing session-oriented parameters that can be used by the Rule Engine and the micro-services:

```c
int msiGetObjType (msiParam_t *objParam, msiParam_t *typeParam, ruleExecInfo_t *rei);
```

```c
int msiReadMDTemplateIntoTagStruct (msiParam_t *bufParam, msiParam_t *tagParam, ruleExecInfo_t *rei);
```

The first step in the implementation is the registration of the new micro-service: **FindObjectType**. The Rule Engine only executes micro-services that are enumerated in the list structure:

```c
microsdef_t  MicrosTable[ ] = {} ;
```

which can be found in the file "server/include/reAction.h". For the micro-service:

```c
int msiGetObjType (msiParam_t *objParam, msiParam_t *typeParam, ruleExecInfo_t *rei);
```

we add the following to the action table.

```c
{"msiGetObjType", 2, (funcPtr) msiGetObjType}
```

This specifies two parameters will be used for the micro-service. We also have to register the second micro-service: **ExtractTag**. For the micro-service:

```c
int msiReadMDTemplateIntoTagStruct (msiParam_t *bufParam, msiParam_t *tagParam, ruleExecInfo_t *rei);
```

we add the following to the action table.

```c
{"msiReadMDTemplateIntoTagStruct", 2, (funcPtr) msiReadMDTemplateIntoTagStruct}
```

This also specifies that two parameters will be used for the micro-service.

The second step of the implementation is the creation of the micro-service: **FindObjectType**. Here we program the code for the micro-service:

```c
int msiGetObjType (msiParam_t *objParam, msiParam_t *typeParam, ruleExecInfo_t *rei)
{
    char*   objName;
    char    objType[MAX_NAME_LEN];
    int     i;
    RE_TEST_MACRO("Looping back on msiGetObjType");
    if (strcmp(objParam->type, STR_MS_T) != 0)
        return(USER_PARAM_TYPE_ERROR);
    objName = (char*) objParam->inpOutStruct;
    i = getObjType (rei->rsComm, objName, objType);
    if (i < 0)   return(i);
    typeParam->inpOutStruct = (char*) strdup(objType);
    typeParam->type = (char*) strdup(STR_MS_T);
    return(0);
}
```
The third step of the implementation is the Doxygen description of the micro-service: **FindObjectType**. We want to provide enough information for users to call the micro-service and for a program to identify it automatically in the future.

```c
/**
fn msiGetObjType (msiParam_t *objParam, msiParam_t *typeParam, ruleExecInfo_t *rei)
@author Arcot Rajasekar
@date 2008-02-01
@brief Finds the type of a given object from the iCAT
@param[in] objParam is a msiParam of type STR_MS_T
@param[out] typeParam is a msiParam of type STR_MS_T
@param[in,out] rei – The RuleExecInfo structure that is automatically handled by the rule engine. The user does not include rei as a parameter in the rule invocation.
@return integer
@return 0 on success
@return USER_PARAM_TYP_ERROR when input param does not match type
@return from getObjType
}@sa getObjType
@post
@pre
**/
```

We repeat the steps for the implementation of the **ExtractTag** micro-service. Here we program the code for the micro-service:

```c
int  msiReadMDTemplateIntoTagStruct (msiParam_t *bufParam, msiParam_t *tagParam, ruleExecInfo_t *rei)
{
  bytesBuf_t  *tmplObjBuf;
tagStruct_t  *tagValues;  /*other internal variables are defined here */
  RE_TEST_MACRO("Looping back on msiReadMDTemplateIntoTagStruct");
  if (strcmp(bufParam->type, BUF_LEN_MS_T) != 0 || bufParam->inpOutBuf == NULL)
    return(USER_PARAM_TYPE_ERROR);
  tmplObjBuf = (bytesBuf_t *) bufParam->inpOutBuf;
tagValues = (tagStruct_t *) mallocAndZero(sizeof(tagStruct_t));
  /* the main code segment that reads the buffer and identifies the <preTag, KeyWord, postTag> triplets goes in here. The triplets are stored in tagValues. */
  if (tagValues->len == 0)
  {
    free(tagValues );
    return(NO_VALUES_FOUND);
  }
tagParam->inOutStruct = (void *) tagValues;
tagParam->type = (char *) strdup(TagStruct_MS_T);
return(0);
}
```

We also describe the micro-service **ExtractTag**. We want to provide enough information for users to call the micro-service and for a program to identify it automatically in the future.

```c
/**
fn msiReadMDTemplateIntoTagStruct(msiParam_t *bufParam, msiParam_t *tagParam, ruleExecInfo_t *rei)
@author Arcot Rajasekar
```
In summary, any function can be converted into a micro-service and made micro-service compliant, with some caveats.

The fourth step consists of testing the micro-services. There are two separate invocation methods that should be tried, from the client-side and from the server-side.

Client-side:

1. Use the irule command.
2. Create a workflow of micro-services.
3. Test with the "loop" functionality.
4. Test with "verbose" functionality.
5. Test without these side-effects.

Server-side:

1. Create an iRODS rule out of the workflow; or,
2. Add the micro-service to an existing rule.

The sequence of micro-services that can be used for testing include:

- msiDataObjOpen: opens an iRODS File
- msiDataObjRead: reads an open iRODS File
- msiReadMDTemplateIntoTagStruct: reads Tag Info into Struct
- msiExtractTemplateMDFromBuf: gets MD using Tag Struct
- msiGetObjType: finds type of object
- msiAssociateKeyValuePairsToObj: ingests extracted metadata
- msiCommit: commit transaction in iCAT
- openObj
- readObj
- getTagsForKV
- getKVPairsUsingTags
- findObjType
The command line workflow can be tested with the following workflow, written in the rule language for iRODS version 3.0:

```
mdExtract {
    openObj(*A, *T_FD);
    getSizeData(*A, *S);
    readObj(*T_FD, *S, *R1_BUF);
    getTagsForKV(*R1_BUF, *TSP);
    openObj(*B, *M_FD);
    readObj(*M_FD, 10000, *R2_BUF);
    getKVPairsUsingTags(*R2_BUF, *TSP, *KVP);
    findObjType(*C, *OTYP);
    ingestBulkMD(*KVP, *C, *OTYP);
    closeObj(*T_FD,*J1);
    closeObj(*M_FD,*J2);
    commitIcat;
}
```

**INPUT** *A="/tempZone/home/rods/Templates/mdTemplate1.txt",
*B="/tempZone/home/rods/test1.email", *C="/tempZone/home/rods/test2.email"


Note that the INPUT variables should be listed on a single line.

The rule used to test the micro-services is very similar to the workflow above:

```
mdExtract(*A,*B,*C) {
    openObj(*A, *T_FD):::closeObj(*T_FD);
    readObj(*T_FD, 10000, *R1_BUF);
    getTagsForKV(*R1_BUF, *TSP):::recover_getTagsForKV(*R1_BUF, *TSP);
    openObj(*B, *M_FD):::closeObj(*M_FD);
    readObj(*M_FD, 10000, *R2_BUF);
    findObjType(*C, *OTYP);
    ingestBulkMD(*KVP, *C, *OTYP):::recover_ingestBulkMD(*KVP, *C, *OTYP);
    closeObj(*T_FD,*J1);
    closeObj(*M_FD,*J2);
    commitIcat:::rollbackIcat;
}
```

3.19 Delaying the Execution of a Micro-Service

We can delay the execution of any micro-service either in the irule execution or in a rule at the server side.

For example, the micro-service `msiSysReplDataObj(*R,*Flag)` replicates an existing iRODS file. In order to delay it we can use:

```
delay("<PLUSET>0000-00-00-00.02.00</PLUSET>")
    {msiSysReplDataObj("tgReplResc");}
```

In a "core.re" file this might be used as follows:
```
acPostProcForPut{
    on($objPath like "/tmpZone/home/tg\*/") {
```
delay("<PLUSET>0000-00-00-00.02.00</PLUSET>") {
    msiSysReplDataObj("tgReplResc");
}

acPostProcForPut {
    on($objPath like "/tmpZone/home/nvo/*") {
        msiSysReplDataObj("nvoReplResc", "null");
    }
}

acPostProcForPut {
}

Three versions of the acPostProcForPut action are listed above. The order is important, as the rule engine will execute the first policy that is satisfied. Thus the last policy should be a generic policy that handles all otherwise non-exceptional cases (it has no condition that must be matched, so it matches everything).

3.20 Summary

We designed and wrote micro-services to extract and ingest template-identified metadata. We coded each module to be micro-service compliant. We tested the micro-services from the command line, and we used the micro-services as a workflow.
PART IV iRODS MICRO-SERVICES

In this chapter:
- Introduction
- Doxygen Output
- Core Micro-services Descriptions
- Module Micro-services Descriptions
- Rule Micro-services Descriptions

The iRODS data grid composes procedures by chaining together micro-services. Information can be passed between micro-services through an in-memory structure called "rei". Each input and output parameter has a well-defined data structure, which is stored in the "rei" structure. Thus, the development of a procedure requires knowledge of the expected data types used by each micro-service.

For each of the micro-services provided in the iRODS release version 3.0, a description of the input and output parameters is provided, along with an example of how the micro-service might be used within a rule. The examples are written using the iRODS 3.0 rule language, a later variant of the "rulegen" language referenced in the iRODS Primer and used up to iRODS 2.5. The text in the examples can be copied into a ".r" file, and then executed using an "irule" command. Further examples can also be found in the code under "/iRODS/clients/icommands/test/rules3.0/". There are several caveats that must be observed:

- Micro-service invocations may be broken across multiple lines in the example because of formatting limitations. These should be combined into a single line that is terminated with the ";" symbol.
- All of the parameters specified in an INPUT line should be combined into a single line. The parameters are separated by commas. Spaces are ignored.
- All of the parameters specified in an OUTPUT line should be combined into a single line.
- All double quotes should be straight quotes, not curly or "smart" (""") quotes. A valid quote example is:
  "Select DATA_ID where DATA_NAME = "*File""
- All single quotes should be straight quotes.

The "rule.r" file can be executed directly by the irule command in iRODS version 3.0:

```
irule -F rule.r
```

Versions 2.5 and earlier of iRODS required that the rule.r file be compiled into a native ".ir" format before execution. If a ".ir" file has been modified to put all strings in double quotes, then version 3.0 can also execute a ".ir" file through the irule command.

The data grid test environment in which the examples execute is assumed to have the following attributes:

- `irodsZone = tempZone`
- `irodsUserName = rods`
- `irodsDefResource = demoResc`

These attributes can be changed by modifying the INPUT parameters for the rule. In addition, the following directories and resources need to be set up to use the rules as listed:

- Test user account with name: testuser
- Test directory at location: /tempZone/home/rods/ruletest
- Test sub-directory at location: /tempZone/home/rods/ruletest/sub
- Test sub-directory at location: /tempZone/home/rods/test
- Test directory at location: /tempZone/home/rods/test
- Test storage resource: testResc
- Test storage group: testgroup (consisting of testResc and demoResc)
A shortened version of the microservice documentation is provided below. To view the full documentation of the latest release, please go online to https://irods.org/doxygen/.

Rules that can be used to try each micro-service are listed in the directory "clients/icommands/test/rules3.0/". The example rule name for a given micro-service can be formed by prepending the micro-service name with "rule" and then appending ".r". Thus the rule that can be used to try the msiGetSystemTime micro-service is "rulemsiGetSystemTime.r".

Rules are also provided for the workflow functions used by the rule language. They are named "ruleworkflow/function.r" where function is replaced by the workflow operator. Thus the rule for illustrating the "if" operator is "ruleworkflowif.r".

Rules are also provided that illustrate additional use cases. They are named:
  rulegenerateBagIt.r
  ruleintegrityACL.r
  ruleintegrityAVU.r,
  ruleintegrityAVUvalue.r
  ruleintegrityExpiry.r
  ruleintegrityFileSize.r
  ruleintegrityFileOwner.r
  listMS.r
  showCore.r

In the following sections, please note that all input parameters for a rule are assumed to be entered on a single line. The code examples may have wrapped due to limitations of the printed text.
4.1 Core :: Collection :: msiAutoReplicateService

```c
msiAutoReplicateService(
    msParam_t * xColl,
    msParam_t * xRecursive,
    msParam_t * xRequireNumReplicas,
    msParam_t * xRescGroup,
    msParam_t * xEmailAccountToNotify
)
```

**Parameters:**

- `[in]` `xColl` - a STR_MS_T containing the collection or object name
- `[in]` `xRecursive` - a STR_MS_T determining whether should be run recursively
  - `true` - will run recursively
  - `false` - default - will not run recursively
- `[in]` `xRequireNumReplicas` - a STR_MS_T specifying the number of required replicas in iRODS. This must be at least 1
- `[in]` `xRescGroup` - a STR_MS_T containing the target resource group name
- `[in]` `xEmailAccountToNotify` - Optional - a STR_MS_T containing the notification email address

**Description:**
This micro-service implements a digital preservation rule that checks data integrity and makes necessary repair(s) as needed when a bad copy is found.

**Note:**
This micro-service is supposed to be run as a periodic service to check if a designated number of required good copies of the dataset(s) from a selected collection are in the system. For a registered copy, it checks if the copy still exits. If the local file is removed by the data owner, the registered copy will be deleted from iRODS. For each replica, whether it is a registered dataset or a vaulted dataset, the service computes the checksum and verifies that the replica is still good. If a bad copy is detected, the copy is deleted. Finally, the service creates necessary replicas to meet the required number of copies.

**Example Usage:**

```c
myTestRule {
    # Input parameters are:
    #  Collection that is being replicated
    #  Recursion flag, set to "true" if enabled
    #  Required number of replicas
    #  Resource group that contains multiple resources
    #  e-mail account which will be notified of success
    # Output from execution of the rule is:
    #  Verify the checksum and number of replicas (1) for collection /tempZone/home/rods/ruletest
    #  E-mail is sent to irod-chat@googlegroups.com
    msiAutoReplicateService(*COLL, "true", *NUMREPLICA, *RESOURCEGROUP, *EMAIL);
    writeLine("stdout","Verify the checksum and number of replicas (*NUMREPLICA) for collection *COLL");
    writeLine("stdout","E-mail is sent to *EMAIL");
}
INPUT *COLL="/tempZone/home/rods/ruletest", *NUMREPLICA="1",
*RESOURCEGROUP="testgroup", *EMAIL="irod-chat@googlegroups.com"
OUTPUT ruleExecOut
```
4.2 Core :: Collection :: msiCollCreate

**msiCollCreate** (msParam_t* inpParam1, msParam_t* inpParam2, msParam_t* outParam)

**Parameters:**

[in] inpParam1 - a CollInp_MS_T or a STR_MS_T which would be taken as dataObj path.

[in] inpParam2 - a STR_MS_T which specifies the flags integer. A flags value of 1 means the parent collections will be created too.

[out] outParam - a INT_MS_T containing the status.

**Description:**

This micro-service creates a new collection by calling rsCollCreate.

**Example Usage:**

```cpp
myTestRule {
    # Input parameters are:
    # Collection that will be created
    # Flag specifying whether to create parent collection
    # Value of 1 means create parent collection
    # Output parameter:
    # Result status for the operation
    # Output from running the example
    # Create collection /tempZone/home/rods/ruletest/sub1
    # Collection created was
    # COLL_NAME = /tempZone/home/rods/ruletest/sub1

    msiCollCreate(*Path,"0",*Status);

    # Verify collection was created
    writeLine("stdout","Create collection *Path");
    writeLine("stdout","Collection created was");
    msiExecStrCondQuery("SELECT COLL_NAME where COLL_NAME = '*Path'", *QOut);
    foreach(*QOut) { msiPrintKeyValPair("stdout",*QOut); }
}
```

**INPUT** *Path="/tempZone/home/rods/ruletest/sub1"

**OUTPUT** ruleExecOut
4.3 Core :: Collection :: msiCollRepl

msiCollRepl ( msParam_t * collection,  
msParam_t * msKeyValStr,  
msParam_t * status )

Parameters:
[in] collection - A CollInp_MS_T or a STR_MS_T with the iRODS path of the collection to replicate.
[in] msKeyValStr - Optional - a STR_MS_T. This is the special msKeyValStr format of keyWd1=value1++++keyWd2=value2++++keyWd3=value3...
If the keyWd is not specified (without the '=' char), the value is assumed to be the target resource ("destRescName") for backward compatibility. Valid keywords are:
- "destRescName" - the target resource to replicate to.
- "backupRescName" - the target resource to backup the data. If this keyWd is used, the backup mode will be switched on.
- "rescName" - the resource of the source copy.
- "updateRepl" - update other replicas with the latest copy. This keyWd has no value. But the '=' character is still needed.
- "replNum" - the replica number to use as source.
- "numThreads" - the number of threads to use.
- "all" - replicate to all resources in the resource group. This keyWd has no value.
- "irodsAdmin" - admin user replicate other users' files. This keyWd has no value.
- "verifyChksum" - verify the transfer using checksum. This keyWd has no value.

[out] status - a CollOprStat_t for detailed operation status.

Description:
This micro-service wraps the rsCollRepl() routine to replicate a collection.

Note:
This call does not require client interaction, which means it can be used through rcExecMyRule (irule) or internally by the server.

Example Usage:

myTestRule {  
# This microservice is deprecated
# Input parameters are:
# Collection that will be replicated, it must contain at least one file
# Target resource in keyword-value form
# Output parameter is:
# Status of operation
# Output from running the example is:
# Replicate collection /tempZone/home/rods/sub1 to location destRescName=testResc

    # Put a file in the collection
    msiDataObjPut(*Path,*Resource,"localPath=*LocalFile++++forceFlag=",*Status);
    msiSplitPath(*Path, *Coll, *File);
    msiCollRepl(*Coll, *RepResource, *status);
    writeLine("stdout","Replicate collection *Coll to location *RepResource");
}
INPUT *RepResource="destRescName=testResc", *Path="/tempZone/home/rods/sub1/foo1", *Resource="demoResc", *LocalFile="foo1"
OUTPUT ruleExecOut
## 4.4 Core :: Collection :: msiDataObjAutoMove

### msiDataObjAutoMove

```c
( msParam_t * inpParam1,
msParam_t * inpParam2,
msParam_t * inpParam3,
msParam_t * inpParam4,
msParam_t * inpParam5 )
```

#### Parameters:

- **inpParam1** - a STR_MS_T containing the object name with path. It usually comes from a query such as "$objPath like '/zone/%%'" in the deployed micro-service.
- **inpParam2** - a STR_MS_T containing the leading collection name to be truncated.
- **inpParam3** - a STR_MS_T containing the destination collection.
- **inpParam4** - a STR_MS_T containing the new owner.
- **inpParam5** - a STR_MS_T containing a flag for whether the checksum should be computed:
  - `true` - default - will compute the checksum.
  - `false` - will not compute the checksum.

#### Description:

This micro-service is used to automatically move a newly created file into a destination collection.

#### Note:

The micro-service moves all replicas of a file in a collection to another user's collection and changes the ownership for the dataset(s) being moved.

#### Example Usage:

```plaintext
myTestRule {
  # Input parameters are:
  # iRODS path of file to move
  # Leading collection name to be truncated from that iRODS path
  # iRODS path for destination collection
  # iRODS owner of the destination collection
  # Flag for whether checksum is computed, value is "true" to compute checksum
  # Output from running the example is:
  # File /tempZone/home/rods/sub1/foo1 is moved to collection /tempZone/home/rods/ruletest/sub1
  # msiSplitPath(*Path, *Coll, *File);
  # msiDataObjAutoMove(*Path, *Coll, *Destination, *Owner, "true");
  # WriteLine("stdout","File *Path is moved to collection *Destination");
}

INPUT *Path="/tempZone/home/rods/sub1/foo1", *Destination="/tempZone/home/rods/ruletest/sub1",
*Owner="rods"
OUTPUT ruleExecOut
```
4.5 Core :: Collection :: msiPhyBundleColl

msiPhyBundleColl ( msParam_t * inpParam1,
                 msParam_t * inpParam2,
                 msParam_t * outParam )

Parameters:
[in] inpParam1 - A StructFileExtAndRegInp_MS_T or a STR_MS_T which would be taken as the collection for the physical bundle.
[in] inpParam2 - optional - a STR_MS_T which specifies the target resource.
[out] outParam - An INT_MS_T containing the status.

Description:
This micro-service bundles a collection into a number of tar files, similar to the iphybun command

Note:
The tar file is written to the /tempZone/bundle/home/rods directory in this example.

Example Usage:

myTestRule {
  # Input parameters are:
  # Collection that will be bundled into a tar file
  # Resource where the tar file will be stored
  # Output parameter is:
  # Status flag for the operation
  # The file is stored under the /tempZone/bundle/home/rods directory in iRODS
  # Output from running the example is
  # Create tar file of collection /tempZone/home/rods/test on resource testResc
  msiPhyBundleColl(*Coll, *Resc, *status);
  writeLine("stdout","Create tar file of collection *Coll on resource *Resc");
}

INPUT *Coll="/tempZone/home/rods/test", *Resc="testResc"
OUTPUT ruleExecOut
4.6  Core :: Collection :: msiReplColl

msiReplColl (    msParam_t * coll
    msParam_t * destRescName,
    msParam_t* options,
    msParam_t * outParam )

Parameters:
[in]  coll - Required - A CollInp_MS_T or a STR_MS_T which would be taken as destination collection path.
[in]  destRescName - A STR_MS_T with the destination resource name.
[in]  options - A STR_MS_T - a group of options in a string delimited by "/%". If the string is empty ("0") or null ("NULL") it will not be used. The options can be the following:
   "all"  replicate to all resources in resource group
   "irodsAdmin"  enables administrator initiated replication
   "backupMode"  if specified, it will try to use 'backup mode' to the destination resource. Means if a good copy already exists in destination resource, it will not throw an error
[out]  outParam - an INT_MS_T containing the status.

Description:
This microservice iterates through a collection and calls rsDataObjRepl to recursively replicate the collection as part of a workflow execution.

Note:
This micro-service is deprecated. Use msiCollRepl instead.

Example Usage:

myTestRule {
  # Input parameters are:
  # Collection path name
  # Destination resource name for replicas
  # Option string containing
  #  all-
  #  irodsAdmin - for administrator initiated replication
  #  backupMode - will not throw an error if a good copy
  #  already exists
  # Output parameter is:
  #  Status
  # Output from running the example is:
  #  Replicate collection /tempZone/home/rods/sub1 to location destRescName=testResc

  # Put a file in the collection
  msiDataObjPut(*Path,*Resource,"localPath=*LocalFile++++forceFlag=",*Status);
  msiSplitPath(*Path, *Coll, *File);

  #Replicate the collection
  msiReplColl(*Coll,*Dest,*Flag,*Status);
  writeLine("stdout","Replicate collection *Coll to location *Dest");
}

INPUT *Path="/tempZone/home/rods/sub1/foo1",*Resource="demoResc", *Dest="testResc",
*LocalFile="foo1", *Flag="backupMode"
OUTPUT ruleExecOut
4.7 Core :: Collection :: msiRmColl

**msiRmColl** (msParam_t * inpParam1, msParam_t * msKeyValStr, msParam_t * outParam)

**Parameters:**

- **[in] inpParam1** - a CollInp_MS_T or a STR_MS_T which would be taken as dataObj path.
- **[in] msKeyValStr** - This is the special msKeyValStr format of keyWd1=value1++++keyWd2=value2++++keyWd3=value3...
  If the keyWd is not specified (without the '=' char), the value is assumed to be one of the keywords listed below for backwards compatibility. Valid keyWds are:
  - "forceFlag" - Remove the data object instead of putting it in the trash. This keyWd has no value. But the '=' character is still needed.
  - "irodsAdminRmTrash" - Admin remove trash. This keyWd has no value.
  - "irodsRmTrash" - Remove trash. This keyWd has no value.
- **[out] outParam** - an INT_MS_T containing the status.

**Description:**
This micro-service calls rsRmColl to recursively remove a collection as part of a workflow execution.

**Example Usage:**

```plaintext
myTestRule {
  # Input parameters are:
  # Collection that will be removed
  # Flag controlling options in the form keyword=value
  # Output parameter is:
  # Status flag for the operation
  # Output from running the example is:
  # Removed collection /tempZone/home/rods/ruletest/sub
  msiRmColl(*Coll,*Flag,*Status);
  writeLine("stdout","Removed collection *Coll");
}

INPUT *Coll="/tempZone/home/rods/ruletest/sub", *Flag="forceFlag=
OUTPUT ruleExecOut
```

68
4.8 Core :: Collection :: msiTarFileCreate

**msiTarFileCreate**

```
msiParam_t* inpParam1,
msiParam_t* inpParam2,
msiParam_t* inpParam3,
msiParam_t* inpParam4
```

**Parameters:**

- **inpParam1** - A StructFileExtAndRegInp_MS_T or a STR_MS_T which would be taken as dataObj path.
- **inpParam2** - A STR_MS_T which specifies the target collection.
- **inpParam3** - optional - A STR_MS_T which specifies the target resource.
- **inpParam4** - optional - A STR_MS_T which specifies if the force flag is set. Set it to "force" if the force option is needed, otherwise no force option will be used.

**Description:**

Creates a tar object file from a target collection

**Note:**

This micro-service calls rsStructFileBundle to create a tar file (inpParam1) from a target collection (inpParam2). The content of the target collection is stored on the physical resource (inpParam3).

**Example Usage:**

```c
myTestRule {
    # Input parameters are:
    # Tar file path name that will be created
    # Collection that will be turned into a tar file
    # Resource where the tar file will be stored
    # Flag controlling options in form keyword=value
    # Output from running the example is:
    # Created tar file /tempZone/home/rods/test/testcoll.tar for collection /tempZone/home/rods/ruletest/sub on resource demoResc
    msiTarFileCreate(*File,*Coll,*Resc,*Flag);
    writeln("stdout","Created tar file *File for collection *Coll on resource *Resc");
}
```

INPUT *File="/tempZone/home/rods/test/testcoll.tar", *Coll="/tempZone/home/rods/ruletest/sub",
*Resc="demoResc", *Flag=""

OUTPUT ruleExecOut
msiTARFileExtract (msParam_t * inpParam1, msParam_t * inpParam2, msParam_t * inpParam3, msParam_t * outParam)

Parameters:
[in] inpParam1 - A StructFileExtAndRegInp_MS_T or a STR_MS_T which would be taken as dataObj path.
[in] inpParam2 - A STR_MS_T which specifies the target collection.
[in] inpParam3 - optional - A STR_MS_T which specifies the target resource.
[out] outParam - An INT_MS_T containing the status.

Description:
Extracts a tar object file into a target collection.

Note:
This micro-service calls rsStructFileExtAndReg to extract a tar file (inpParam1) into a target collection (inpParam2). The content of the target collection is stored on the physical resource (inpParam3).

Example Usage:

myTestRule {
  # Input parameters are:
  # Tar file within iRODS that will have its files extracted
  # Collection where the extracted files will be placed
  # Resource where the extracted files will be written
  # Output parameter:
  # Status flag for the operation
  # Output from running the example is:
  # Extract files from a tar file into collection /tempZone/home/rods/ruletest/sub on resource demoResc
  msiTarFileExtract(*File,*Coll,*Resc,*Status);
  writeLine("stdout","Extract files from a tar file *File into collection *Coll on resource *Resc");
}

INPUT *File="/tempZone/home/rods/test/testcoll.tar", *Coll="/tempZone/home/rods/ruletest/sub", *Resc="demoResc"
OUTPUT ruleExecOut
4.10 Core :: Database Object :: msiDboExec

```c
msiDboExec (msParam_t * dbrName, msParam_t * dboName, msParam_t * dborName, msParam_t * options, msParam_t * inpParam1, msParam_t * inpParam2, msParam_t * inpParam3, msParam_t * inpParam4, msParam_t * inpParam5, msParam_t * inpParam6)
```

**Parameters:**
- **[in] dbrName**: a STR_MS_T, name of the DBR being used
- **[in] dboName**: a STR_MS_T, name of the DBO being used
- **[in] dborName**: a STR_MS_T, name of the database object record to be created
- **[in] options**: a STR_MS_T, currently 'force' or not for writing dborName
- **[in] inpParam1**: Optional - STR_MS_T parameters to the DBO SQL.
- **[in] inpParam2**: Optional - STR_MS_T parameters to the DBO SQL.
- **[in] inpParam3**: Optional - STR_MS_T parameters to the DBO SQL.
- **[in] inpParam4**: Optional - STR_MS_T parameters to the DBO SQL.
- **[in] inpParam5**: Optional - STR_MS_T parameters to the DBO SQL.
- **[in] inpParam6**: Optional - STR_MS_T parameters to the DBO SQL.

**Description:**
Execute a database object on a DBR

**Note:**
To setup a database resource, make an entry to server/config/dbr.config for the new DBR. This has the DBR-name, DBMS username and password, and the DB type (postgresql, oracle, mysql).
Add an entry to ~/.odbc.ini. Use the name: IRODS_DBR_dbrname for the entry. For example for a DBR called dbr2, it would be IRODS_DBR_dbr2.
Enable DBR compile-time settings and rebuild. Edit config/config.mk, setting DBR, DBR_TYPE, and either postgres, oracle, or mysql parameters.
Use iadmin to define the DBR resource in the ICAT.
```bash
iadmin mkresc DBR2 database postgresql localhost
```
Give access permission
```bash
ichmod -R own rods dbr2
```
Create and store a database object which contains the SQL command

**Example Usage:**
```bash
myTestRule {
# Input parameters are:
# Database resource name
# Database object name
# Database object record name if the result is being saved
# Force flag for database object record
# 6 additional optional parameters used as input to the database object
# Contents of the DBO are:
#  select user_name, zone_name from r_user_main where zone_name = ?
# Output from running the example is:
# Executed the following SQL command
# <dbr>dbr2</dbr>
# <sql>select user_name, zone_name from r_user_main where zone_name = ?</sql>
```
```xml
    # <arg>tempZone</arg>
    # <column_descriptions>
    #  user_name|zone_name|
    # </column_descriptions>
    # <rows>
    #  rodsadmin|tempZone|
    #  rodsBoot|tempZone|
    #  public|tempZone|
    #  rods|tempZone|
    #  testuser|tempZone|
    # </rows>
    writeLine("stdout","Executed the following SQL command");
    msiDboExec(*DBR,*DBO,*DBOR,*Flag,*Inp1,*Inp2,*Inp3,*Inp4,*Inp5,*Inp6);
```

```c
INPUT *DBR="dbr2", *DBO="/tempZone/home/rods/DBOtest/lt1.pg", *DBOR="", *Flag="force",
       *Inp1="tempZone", *Inp2="", *Inp3="", *Inp4="", *Inp5="", *Inp6=""
OUTPUT ruleExecOut
```
4.11 Core :: Database Object :: msiDbrCommit

**msiDbrCommit** (msParam_t * dbrName)

**Parameters:**
[in] dbrName - a STR_MS_T, name of the DBR being used

**Description:**
This micro-service does a commit on a Database Resource (DBR).

**Note:**
To setup a database resource, make an entry to server/config/dbr.config for the new DBR. This has the DBR-name, DBMS username and password, and the DB type (postgresql, oracle, mysql).
Add an entry to ~/.odbc.ini. Use the name: IRODS_DBR_dbrname for the entry. For example for a DBR called dbr2, it would be IRODS_DBR_dbr2.
Enable DBR compile-time settings and rebuild. Edit config/config.mk, setting DBR, DBR_TYPE, and either postgres, oracle, or mysql parameters.
Use iadmin to define the DBR resource in the ICAT.
    iadmin mkresc DBR2 database postgresql localhost
Give access permission
    ichmod -R own rods dbr2
Create and store a database object which contains the SQL command

**Example Usage:**

```plaintext
myTestRule {
# Input parameters are:
# Database resource name
# Output from running the example is:
# Changes to database resource dbr2 were committed
  msiDbrCommit(*DBR);
  writeLine("stdout","Changes to database resource *DBR were committed");
}
INPUT *DBR="dbr2"
OUTPUT ruleExecOut
```
4.12 Core :: Database Object :: msiDbrRollback

msiDbrRollback( msParam_t * dbrName )

Parameters:
[in] dbrName - a STR_MS_T, name of the DBR being used

Description:
This micro-service does a rollback on a Database Resource (DBR)

Note:
To setup a database resource, make an entry to server/config/dbr.config for the new DBR. This has the DBR-name, DBMS username and password, and the DB type (postgresql, oracle, mysql).
Add an entry to ~/.odbc.ini. Use the name: IRODS_DBR_dbrname for the entry. For example for a DBR called dbr2, it would be IRODS_DBR_dbr2.
Enable DBR compile-time settings and rebuild. Edit config/config.mk, setting DBR, DBR_TYPE, and either postgres, oracle, or mysql parameters.
Use iadmin to define the DBR resource in the ICAT.

Example Usage:

myTestRule {
    # Input parameters are:
    # Database resource name
    # Output from running the example is:
    # Changes to database resource dbr2 were rolled back
    msiDbrRollback(*DBR);
    writeLine("stdout","Changes to database resource *DBR were rolled back");
}
INPUT *DBR="dbr2"
OUTPUT ruleExecOut
4.13  Core :: Data Object Low-level :: msiDataObjClose

`msiDataObjClose`  
`msParam_t * inpParam,`  
`msParam_t * outParam`  

**Parameters:**
[in]  `inpParam` - `inpParam` is a msParam of type INT_MS_T or STR_MS_T.
[out] `outParam` - `outParam` is a msParam of type INT_MS_T.

**Description:**
This micro-service performs a low-level close for an opened/created data object.

**Note:**
Can be called by client through irule.

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  # Path
  # Flags specifying resource, and force option in format keyword=value
  # Output parameter is:
  # File descriptor for the file
  # Output from running the example is
  #  Created and closed file /tempZone/home/rods/test/foo4
  msiDataObjCreate(*ObjB,*OFlagsB,*D_FD);
  writeLine("stdout","Created and closed file *ObjB");
  msiDataObjClose(*D_FD,*Status2);
}
```

Example:

```
INPUT *Resc="demoResc", *ObjB="/tempZone/home/rods/test/foo4",
*OFlagsB="destRescName=demoResc++++forceFlag="
OUTPUT ruleExecOut
```
4.14 Core :: Data Object Low-level :: msiDataObjCreate

msiDataObjCreate

( msParam_t * inpParam1,
  msParam_t * msKeyValStr,
  msParam_t * outParam )

Parameters:

[in] inpParam1 - A DataObjInp_MS_T or STR_MS_T which would be taken as dataObj path.

[in] msKeyValStr - Optional - a STR_MS_T. This is the special msKeyValStr format of keyWd1=value1++++keyWd2=value2++++keyWd3=value3... If the keyWd is not specified (without the '=' char), the value is assumed to be the target resource ("destRescName") for backward compatibility. Valid keyWds are:
- "destRescName" - the target resource.
- "forceFlag" - overwrite existing copy. This keyWd has no value. But the '=' character is needed.
- "createFlag" - the file mode of the data object.
- "dataType" - the data type of the data object.
- "dataSize" - the size of the data object. This input is optional.

[out] outParam - a INT_MS_T containing the descriptor of the create.

Description:
Creates a file descriptor for a data object, for subsequent reading or writing.

Note:
none

Example Usage:

myTestRule {
  # Input parameters are:
  # Path
  # Flags specifying resource, and force option in format keyword=value
  # Output parameter is:
  # File descriptor for the file
  # Output from running the example is
  # Created and closed file /tempZone/home/rods/test/foo4
  msiDataObjCreate(*ObjB,*OFlagsB,*D_FD);
  msiDataObjClose(*D_FD,*Status2);
  writeLine("stdout","Created and closed file *ObjB");
}

INPUT *Resc="demoResc", *ObjB="/tempZone/home/rods/test/foo4",
  *OFlagsB="destRescName=demoResc++++forceFlag="
OUTPUT ruleExecOut
4.15 Core :: Data Object Low-level :: msiDataObjLseek

`msiDataObjLseek` (msParam_t * inpParam1, msParam_t * inpParam2, msParam_t * inpParam3, msParam_t * outParam)

**Parameters:**

- **[in]** `inpParam1` - a msParam of type DataObjLseekInp_MS_T or INT_MS_T or a STR_MS_T which would be the file descriptor.
- **[in]** `inpParam2` - Optional - a msParam of type DOUBLE_MS_T or a STR_MS_T which would be the offset.
- **[in]** `inpParam3` - Optional - a msParam of type INT_MS_T or a STR_MS_T which would be location for offset. Can be SEEK_SET, SEEK_CUR, and SEEK_END.
- **[out]** `outParam` - a msParam of type Double_MS_T or DataObjLseekOut_MS_T which is the return status.

**Description:**
This a micro-service that performs a low-level (file) seek of an opened data object.

**Note:**
Can be called by client through irule

**Example Usage:**

```c
myTestRule {
    # Input parameters are:
    # File descriptor
    # Optional Offset from specified location
    # Optional location for offset: SEEK_SET, SEEK_CUR, and SEEK_END
    # Output Parameter is:
    # Status of operation
    # Output from running the example is:
    # Open file /tempZone/home/rods/test/foo1, create file /tempZone/home/rods/test/foo4, copy 100 bytes starting at location 10
    msiDataObjOpen(*OFlags,*S_FD);
    msiDataObjCreate(*ObjB,*OFlagsB,*D_FD);
    msiDataObjLseek(*S_FD,*Offset,*Loc,*Status1);
    msiDataObjRead(*S_FD,*Len,*R_BUF);
    msiDataObjWrite(*D_FD,*R_BUF,*W_LEN);
    msiDataObjClose(*S_FD,*Status2);
    msiDataObjClose(*D_FD,*Status3);
    writeLine("stdout","Open file *Obj, create file *ObjB, copy *Len bytes starting at location *Offset");
}
```

INPUT *Obj="/tempZone/home/rods/test/foo1", *OFlags="objPath=/tempZone/home/rods/test/foo1++++rescName=demoResc++++replNum=0++++openFlags=O_RDONLY", *ObjB="/tempZone/home/rods/test/foo4", *OFlagsB="destRescName=demoResc++++forceFlag=",*Offset="10", *Loc="SEEK_SET", *Len="100"

OUTPUT ruleExecOut
4.16 Core :: Data Object Low-level :: msiDataObjOpen

msiDataObjOpen (msParam_t * inpParam, msParam_t * outParam)

Parameters:
[in] inpParam - a msParam of type DataObjInp_MS_T or a STR_MS_T which would be taken as msKeyValStr. msKeyValStr - This is the special msKeyValStr format of
keyWd1=value1++++keyWd2=value2++++keyWd3=value3...
If the keyWd is not specified (without the '=' char), the value is assumed to be the path of the data object("objPath") for backward compatibility. Valid keyWds are:
"objPath" - the path of the data object to open.
"rescName" - the resource of the data object to open.
"replNum" - the replica number of the copy to open.
"openFlags" - the open flags. Valid open flags are: O_RDONLY, O_WRONLY, O_RDWR and O_TRUNC. These can be combined by concatenation, e.g. O_WRONLYO_TRUNC (without the '|' character). The default open flag is O_RDONLY.

[out] outParam - a msParam of type INT_MS_T containing the descriptor of the opened file.

Description:
This micro-service performs a low-level open for an existing data object

Note:
Can be called by client through irule

Example Usage:

myTestRule {
  # Input parameters are:
  # File descriptor
  # Optional length to read
  # Output parameter is:
  # Buffer holding the data read
  # Output from running the example is:
  # Open file /tempZone/home/rods/test/foo1, create file /tempZone/home/rods/test/foo4, copy 100 bytes starting at location 10
  msiDataObjOpen(*OFlags,*S_FD);
  msiDataObjCreate(*ObjB,*OFlagsB,*D_FD);
  msiDataObjLseek(*S_FD,*Offset,*Loc,*Status1);
  msiDataObjRead(*S_FD,*Len,*R_BUF);
  msiDataObjWrite(*D_FD,*R_BUF,*W_LEN);
  msiDataObjClose(*D_FD,*Status2);
  msiDataObjClose(*D_FD,*Status3);
  writeLine("stdout","Open file *Obj, create file *ObjB, copy *Len bytes starting at location *Offset");
}

INPUT *Obj="/tempZone/home/rods/test/foo1",
*OFlags="objPath=/tempZone/home/rods/test/foo1++++rescName=demoResc++++replNum=0++++open
Flags=O_RDONLY", *ObjB="/tempZone/home/rods/test/foo4",
*OFlagsB="destRescName=demoResc++++forceFlag=no", *Offset="10", *Loc="SEEK_SET", *Len="100"
OUTPUT ruleExecOut
4.17 Core :: Data Object Low-level :: msiDataObjRead

msiDataObjRead ( msParam_t * inpParam1,
                  msParam_t * inpParam2,
                  msParam_t * outParam )

Parameters:
[in] inpParam1 - a msParam of type DataObjReadInp_MS_T or INT_MS_T or STR_MS_T which would be the file descriptor.
[in] inpParam2 - Optional - a msParam of type INT_MS_T or STR_MS_T which would be the length.
[out] outParam - a msParam of type BUF_LEN_MS_T.

Description:
This micro-service performs a low-level read of an opened data object.

Note:
Can be called by client through irule.

Example Usage:

myTestRule {
  # Input parameters are:
  # File descriptor
  # Optional length to read
  # Output Parameter is:
  # Buffer holding the data read
  # Output from running the example is:
  # Open file /tempZone/home/rods/test/foo1, create file /tempZone/home/rods/test/foo4, copy 100 bytes starting at location 10
  msiDataObjOpen(*OFlags,*S_FD);
  msiDataObjCreate(*ObjB,*OFlagsB,*D_FD);
  msiDataObjLseek(*S_FD,*Offset,*Loc,*Status1);
  msiDataObjRead(*S_FD,*Len,*R_BUF);
  msiDataObjWrite(*D_FD,*R_BUF,*W_LEN);
  msiDataObjClose(*S_FD,*Status2);
  msiDataObjClose(*D_FD,*Status3);
  writeLine("stdout","Open file *Obj, create file *ObjB, copy *Len bytes starting at location *Offset to *ObjB");
}

INPUT *Nu=", *Obj="/tempZone/home/rods/test/foo1", *Resc="demoResc", *Repl="0",
*Flag="O_RDONLY",
*OFlags="objPath=*Obj++++rescName=*Resc++++replNum=*Repl++++openFlags=*Flag",
*ObjB="/tempZone/home/rods/test/foo4", *OFlagsB="destRescName=*Resc++++forceFlag=*Nu",
*Offset=10, *Loc="SEEK_SET", *Len=100
OUTPUT ruleExecOut
Core :: Data Object Low-level :: msiDataObjWrite

**msiDataObjWrite**

( msParam_t * inpParam1,  
  msParam_t * inpParam2, 
  msParam_t * outParam )

**Parameters:**

- **inpParam1**
  - A msParam of type DataObjWriteInp_MS_T or INT_MS_T or a STR_MS_T which would be the file descriptor.

- **inpParam2**
  - Optional - A msParam of type BUF_LEN_MS_T or a STR_MS_T, the input can be an inpOutBuf from a previous read. "stderr", "stdout" can be passed as well.

- **outParam**
  - A msParam of type INT_MS_T for the length written.

**Description:**
This micro-service performs a low-level write to an opened data object.

**Note:**
Can be called by client through irule.

**Example Usage:**

```c
myTestRule {  
  # Input parameters are:
  #   File descriptor
  #   Buffer that is being written
  # Output parameter is:
  #   Length that is written
  # Output from running the example is:
  #   Open file /tempZone/home/rods/test/foo1, create file /tempZone/home/rods/test/foo4, copy 100 bytes starting at location 10
  msiDataObjOpen(*OFlags,*S_FD);
  msiDataObjCreate(*ObjB,*OFlagsB,*D_FD);
  msiDataObjLseek(*S_FD,*Offset,*Loc,*Status1);
  msiDataObjRead(*S_FD,*Len,*R_BUF);
  msiDataObjWrite(*D_FD,*R_BUF,*W_LEN);
  msiDataObjClose(*S_FD,*Status2);
  msiDataObjClose(*D_FD,*Status3);
  writeLine("stdout","Open file *Obj, create file *ObjB, copy *Len bytes starting at location *Offset");
}
```

**INPUT**

- *Obj="/tempZone/home/rods/test/foo1",
- *OFlags="objPath=/tempZone/home/rods/test/foo1++++rescName=demoResc++++replNum=0++++openFlags=O_RDONLY", *ObjB="/tempZone/home/rods/test/foo4",
- *OFlagsB="destRescName=demoResc++++forceFlag=",
- *Offset="10", *Loc="SEEK_SET", *Len="100"

**OUTPUT**

ruleExecOut
4.19 Core :: Data Object :: msiCheckAccess

msiCheckAccess(
    msParam_t *inObjName,
    msParam_t *inOperation,
    msParam_t *outResult
)

Parameters:
[in] inObjName - a msParam of type STR_MS_T with the object name
[in] inOperation - a msParam of type STR_MS_T with the type of desired access
[out] outResult - a msParam of type STR_MS_T for the result of the check,
                   with 0 for failure and 1 for success

Description:
This micro-service checks whether the desired access is permitted.

Note:
The micro-service relies upon session variables defined for user access to define whose permissions are
being checked. Only the access permissions of the person executing the micro-service can be checked. See
ruleintegrityFileOwner.r for how to check access permission for any person. The following types of
hierarchical access can be checked. The list is ordered from lowest to highest access permission. A
higher access permission grants access to all lower access permissions.

null
execute
read annotation
read system metadata
read object
write annotation
create metadata
modify metadata
administer object
create object
modify object
delete object
create token
delete token
curate
own

Example Usage:

myTestRule { 
# Input parameters are:
# Name of object
# Access permission that will be checked
# Output parameter is:
# Result, 0 for failure and 1 for success
msiCheckAccess(*Path,*Acl,*Result);
if(*Result == 0) {
    writeLine("stdout","File *Path does not have access *Acl"); }
else {writeLine("stdout","File *Path has access *Acl"); }
}
INPUT *Path = "/tempZone/home/rods/sub1/foo1", *Acl = "own"
OUTPUT ruleExecOut
Core :: Data Object :: msiCheckOwner

msiCheckOwner( )

Parameters:
None.

Description:
This micro-service checks whether the user is the owner for a file operation.

Note:
This micro-service can only be used within the "core.re" file for policies that have the S3 session variable $userNameClient set as defined in the iRODS Primer.

Example Usage:

acPostProcForPut {
  # The msiCheckOwner microservice reads the data object rei structure
  # and can only be use with policies that set the S3 session variables
  # Input parameter is:
  # None
  # Output parameter is:
  # None
  # Output from running the example is:
  # Username is rods
    ON (msiCheckOwner==0) {
      writeLine("stdout","Username is $userNameClient");
    }
}
4.21  Core :: Data Object :: msiCollRsync

msiCollRsync (  
    msParam_t*  inpParam1,  
    msParam_t*  inpParam2,  
    msParam_t*  inpParam3,  
    msParam_t*  inpParam4,  
    msParam_t*  outParam  )

Parameters:
[in]  inpParam1  -  a STR_MS_T which specifies the source collection path.
[in]  inpParam2  -  a STR_MS_T which specifies the target collection path.
[in]  inpParam3  -  Optional  -  a STR_MS_T which specifies the target resource.
[in]  inpParam4  -  Optional  -  a STR_MS_T which specifies the rsync mode  
    (RSYNC_MODE_KW).  Valid mode is IRODS_TO_IRODS.
[out]  outParam  -  a INT_MS_T containing the status.

Description:
This micro-service recursively syncs a source collection to a target collection.

Example Usage:

myTestRule {  
    # Input parameters are:
    #  Source collection path
    #  Target collection path
    #  Optional target resource
    #  Optional synchronization mode:  IRODS_TO_IRODS
    # Output parameter is:
    #  Status of the operation
    # Output from running the example is:
    #  Synchronized collection /tempZone/home/rods/sub1 with collection /tempZone/home/rods/sub2
    msiCollRsync(*srcColl,*destColl,*Resource,"IRODS_TO_IRODS",*Status);
    writeLine("stdout","Synchronized collection *srcColl with collection *destColl");  
    }  

INPUT *srcColl="/tempZone/home/rods/sub1", *destColl="/tempZone/home/rods/sub2",  
    *Resource="demoResc"
OUTPUT ruleExecOut
Core :: Data Object :: msiDataObjChksum

**msiDataObjChksum** (msParam_t * inpParam1,
msParam_t * msKeyValStr,
msParam_t * outParam )

**Parameters:**
- **[in]** inpParam1 - A DataObjInp_MS_T or a STR_MS_T which would be taken as dataObj path.
- **[in]** msKeyValStr - Optional - a STR_MS_T. This is the special msKeyValStr format of keyWd1=value1++++keyWd2=value2++++keyWd3=value3...
  If the keyWd is not specified (without the '=' char), the value is assumed to be the target resource ("destRescName") for backward compatibility. Valid keyWds are:
  - "ChksumAll" - checksum all replicas. This keyWd has no value. But the '=' character is still needed.
  - "verifyChksum" - verify the chksum value.
  - "forceChksum" - checksum data-objects even if a checksum already exists in iCAT. This keyWd has no value.
  - "replNum" - the replica number to checksum. This keyWd has no value.
- **[out]** outParam - a STR_MS_T containing the checksum value.

**Description:**
This micro-service calls rsDataObjChksum to checksum the input data object as part of a workflow execution. The example generates checksums for all replicas.

**Example Usage:**

myTestRule {
  # Input parameters are:
  # Data object path
  # Optional flags in form Keyword=value
  #  ChksumAll=
  #  verifyChksum=
  #  forceChksum=
  #  replNum=
  # Output parameters are:
  #  Checksum value
  # Output from running the example is
  # Collection is /tempZone/home/rods/sub1 and file is foo1
  # Saved checksum for file foo1 is f03e80c9994d137614935e4913e53417, new checksum is f03e80c9994d137614935e4913e53417
  msiSplitPath(*dataObject,*Coll,*File);
  writeLine("stdout","Collection is *Coll and file is *File");
  msiMakeGenQuery("DATA_CHECKSUM","DATA_NAME = '*File' AND COLL_NAME = '*Coll'",*GenQInp);
  msiExecGenQuery(*GenQInp,*GenQOut);
  foreach(*GenQOut) {
    msiGetValByKey(*GenQOut,"DATA_CHECKSUM",*chkSumS);
    msiDataObjChksum(*dataObject,*Flags,*chkSum);
    writeLine("stdout","Saved checksum for file *File is *chkSumS, new checksum is *chkSum");
  }
}

INPUT *dataObject="/tempZone/home/rods/sub1/foo1", *Flags="forceChksum="
OUTPUT ruleExecOut
Core :: Data Object :: msiDataObjChksumWithOptions

$\texttt{msiDataObjChksumWithOptions}(\text{msParam}_t^* \text{inpParam1},$
\text{msParam}_t^* \text{inpParam2},$
\text{msParam}_t^* \text{inpParam3},$
\text{msParam}_t^* \text{outParam} )$

**Parameters:**

- [in] inpParam1 - a msParam of type DataObjInp_MS_T or a STR_MS_T which would be taken as dataObj path.
- [in] inpParam2 - Optional - a STR_MS_T which specifies "verifyChksum" (VERIFY_CHKSUM_KW) or "forceChksum" (FORCE_CHKSUM_KW).
- [in] inpParam3 - Optional - a STR_MS_T which specifies the "ChksumAll" (CHKSUM_ALL_KW) or a INT which gives the replica number.
- [out] outParam - a STR_MS_T containing the chksum value.

**Description:**

This micro-service calls rsDataObjChksum to checksum the input data object as part of a workflow execution, with options input as separate parameters.

**Note:**

This micro-service is deprecated. Since version 2.2, msiDataObjChksum can take a parameter with key-values format which makes this micro-service obsolete. Can be called by client through irule.

**Example Usage:**

```irule
myTestRule {}  
# This example is deprecated  
# Input parameters are  
# Data object path  
# Optional flag for verifyChksum or forceChksum  
# Optional flag for ChksumAll or an integer for the replica number  
# Output parameter is:  
# Checksum value  
# Output from running the example is:  
# Collection is /tempZone/home/rods/sub1 and file is foo1  
# Saved checksum for file foo1 is f03e80c9994d137614935e4913e53417, new checksum is f03e80c9994d137614935e4913e53417  
# msiSplitPath(*Path,*Coll,*File);  
# msiMakeGenQuery("DATA_CHECKSUM","DATA_NAME = '*File' AND COLL_NAME = '*Coll'",*GenQInp);  
# foreach(*GenQOut) {  
# msiGetValByKey(*GenQOut,"DATA_CHECKSUM",*chkSumS);  
# msiDataObjChksumWithOptions(*Path,*Flags,0,*chkSum);  
# }  
# OUTPUT ruleExecOut
INPUT *Path="/tempZone/home/rods/sub1/foo1", *Flags="forceChksum"
```

85
4.24 Core :: Data Object :: msiDataObjCopy

msiDataObjCopy ( msParam_t * inpParam1, msParam_t * inpParam2, msParam_t * msKeyValStr, msParam_t * outParam )

Parameters:
[in] inpParam1 - a DataObjCopyInp_MS_T or DataObjInp_MS_T which is the source DataObjInp or STR_MS_T which would be the source object path.
[in] inpParam2 - Optional - a DataObjInp_MS_T which is the destination DataObjInp or STR_MS_T which would be the destination object path.
[in] msKeyValStr - Optional - a STR_MS_T. This is the special msKeyValStr format of keyWd1=value1++++keyWd2=value2++++keyWd3=value3...
If the keyWd is not specified (without the '=' char), the value is assumed to be the target resource ("destRescName") for backward compatibility. Valid keyWds are:
"destRescName" - the resource to copy to.
"forceFlag" - overwrite existing copy. This keyWd has no value.
But the '=' character is still needed.
"numThreads" - the number of threads to use.
"filePath" - The physical file path of the uploaded file on the server.
"dataType" - the data type of the file.
"verifyChksum" - verify the transfer using checksum. this keyWd has no value.
But the '=' character is still needed.
[out] outParam - a INT_MS_T for the status.

Description:
This micro-service copies a file from one logical (source) collection to another logical (destination) collection. The destination collection can be put on another storage resource.

Example Usage:

myTestRule {
# Input parameters are:
# Source data object path
# Optional destination object path
# Optional flags in form keyword=value
# destRescName
# forceFlag=
# numThreads
# filePath="Physical file path of the uploaded file on the server"
# dataType
# verifyChksum=
# Output parameter is:
# Status
# Output from running the example is
# File /tempZone/home/rods/sub1/foo1 copied to /tempZone/home/rods/sub2/foo1
  msiDataObjCopy(*SourceFile,*DestFile,"forceFlag=",*Status);
  writeLine("stdout","File *SourceFile copied to *DestFile");
}
INPUT *SourceFile="/tempZone/home/rods/sub1/foo1", *DestFile="/tempZone/home/rods/sub2/foo1"
OUTPUT ruleExecOut
4.25 Core :: Data Object :: msiDataObjGet

```
msiDataObjGet(msParam_t * inpParam1,
               msParam_t * msKeyValStr,
               msParam_t * outParam)
```

**Parameters:**

- **inpParam1** - A DataObjInp_MS_T or STR_MS_T which would be taken as dataObj path.
- **msKeyValStr** - Optional - a STR_MS_T. This is the special msKeyValStr format of keyWd1=value1++++keyWd2=value2++++keyWd3=value3...
  If the keyWd is not specified (without the '=' char), the value is assumed to be the client's local file path ("localPath") for backward compatibility. Valid keyWds are:
  - "localPath" - the client's local file path.
  - "rescName" - the resource of the copy to get.
  - "replNum" - the replica number of the copy to get.
  - "numThreads" - the number of threads to use.
  - "forceFlag" - overwrite local copy. This keyWd has no value. But the '=' character is still needed
  - "verifyChksum" - verify the transfer using checksum. This keyWd has no value. But the '=' character is still needed.

- **outParam** - a INT_MS_T containing the status.

**Description:**

This micro-service gets a data object by requesting the client to call a rcDataObjGet API as part of a workflow execution.

**Note:**

This call should only be used through the rcExecMyRule (irule) call i.e., rule execution initiated by clients and should not be called internally by the server since it interacts with the client through the normal client/server socket connection. Also, it should never be called through delay since it requires client interaction. The localPath is required on input.

**Example Usage:**

```
myTestRule {
# Input parameters are:
#   Data object path
#   Flags in form keyword=value
#   localPath
#   rescName
#   replNum
#   numThreads
#   forceFlag
#   verifyChksum
# Output parameter is
#   Status
# Output from running the example is:
#   File /tempZone/home/rods/sub1/foo1 is retrieved from the data grid
#   msiSplitPath(*SourceFile,*Coll,*File);
#   msiDataObjGet(*SourceFile,"localPath=./*File++++forceFlag=",*Status);
#   writeLine("stdout","File *SourceFile is retrieved from the data grid");
}

INPUT *SourceFile="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
```
4.26  Core :: Data Object :: msiDataObjGetWithOptions

msiDataObjGetWithOptions ( msParam_t * inpParam1,
                         msParam_t * inpParam2,
                         msParam_t * srcrescParam,
                         msParam_t * outParam )

Parameters:
[in]  inpParam1  -  A DataObjInp_MS_T or STR_MS_T which would be taken as dataObj path.
[in]  inpParam2  -  Optional - a STR_MS_T which specifies the client's local file path.
[in]  srcrescParam  -  Optional - a STR_MS_T which specifies the source resource.
[out] outParam  -  a INT_MS_T containing the status.

Description:
This micro-service requests the client to call a rcDataObjGet API as part of a workflow execution, with
options input as separate parameters.

Note:
This micro-service is deprecated, and is replaced with msiDataObjGet. This call should only be used
through the rcExecMyRule (irule) call i.e., rule execution initiated by clients and should not be called
internally by the server since it interacts with the client through the normal client/server socket connection.
Also, it should never be called through delay since it requires client interaction. The example assumes that
a local copy of the file is not already present on the local resource.

Example Usage:

myTestRule {
  # Input parameters are:
  #   Data object path
  #   Local file path
  #   Source resource
  # Output parameter is:
  #   Status
  # Output from executing the example is:
  #   File /tempZone/home/rods/sub1/foo1 is retrieved from the data grid and written to foo1
  msiSplitPath(*SourceFile,*Coll,*File);
  msiDataObjGetWithOptions(*SourceFile,".*File","*Resource,*Status);
  writeLine("stdout","File *SourceFile is retrieved from the data grid and written to *File");
}

INPUT *SourceFile="/tempZone/home/rods/sub1/foo1", *Resource="demoResc"
OUTPUT ruleExecOut
4.27 Core :: Data Object :: msiDataObjPhymv

msiDataObjPhymv (msParam_t * inpParam1,
msParam_t * inpParam2,
msParam_t * inpParam3,
msParam_t * inpParam4,
msParam_t * inpParam5,
msParam_t * outParam )

Parameters:
[in] inpParam1 - A DataObjInp_MS_T or STR_MS_T which would be taken as dataObj path.
[in] inpParam2 - Optional - a STR_MS_T which specifies the dest resourceName.
[in] inpParam3 - Optional - a STR_MS_T which specifies the src resourceName.
[in] inpParam4 - Optional - a STR_MS_T which specifies the replNum.
[in] inpParam5 - Optional - a STR_MS_T which specifies the IRODS_ADMIN_KW, irodsAdmin, for administrator controlled data movement
[out] outParam - a INT_MS_T containing the status.

Description:
This micro-service calls rsDataObjPhymv to physically move the input data object to another resource.

Note:
If the policy acSetRescSchemeForCreate sets a default resource as forced, the physical move will not be done to the requested resource.

Example Usage:

myTestRule {
# Input parameters are:
# Data object path
# Optional destination resource name
# Optional source resource name
# Optional replica number
# Optional keyword for IRODS_ADMIN
# Output parameters are:
# Status
# Output from running the example is:
# Replica number 0 of file /tempZone/home/rods/sub1/foo1 is moved from resource demoResc to resource testResc
msiDataObjPhymv(*SourceFile,*DestResource,*SourceResource,*ReplicaNumber,"null",*Status);
writeLine("stdout","Replica number *ReplicaNumber of file *SourceFile is moved from resource *SourceResource to resource *DestResource");
}

INPUT *SourceFile="/tempZone/home/rods/sub1/foo1", *DestResource="testResc",
*SourceResource="demoResc", *ReplicaNumber="0"
OUTPUT ruleExecOut
msiDataObjPut (msParam_t * inpParam1,  
               msParam_t * inpParam2,  
               msParam_t * msKeyValStr,  
               msParam_t * outParam)

Parameters:
[in] inpParam1 - A DataObjInp_MS_T or STR_MS_T which would be taken as dataObj path.
[in] inpParam2 - Optional - a STR_MS_T which specifies the resource.
[in] msKeyValStr - Optional - a STR_MS_T. This is the special msKeyValStr format of
keyWd1=value1+++++keyWd2=value2+++++keyWd3=value3...
If the keyWd is not specified (without the '=' char), the value is
assumed to be the client's local file path ("localPath") for
backward compatibility. Valid keyWds are:
"localPath" - the client's local file path.
"destRescName" - the target resource - where the object should go.
"all" - upload to all resources
"forceFlag" - overwrite existing copy. This keyWd has
no value. But the '=' character is still needed.
"replNum" - the replica number to overwrite.
"numThreads" - the number of threads to use.
"filePath" - The physical file path of the uploaded file on the server.
"dataType" - the data type of the file.
"verifyChksum" - verify the transfer using checksum. This keyWd has no
value. But the '=' character is still needed.
[out] outParam - a INT_MS_T containing the status.

Description:
This micro-service requests the client to call a rcDataObjPut API as part of a workflow execution.

Note:
This call should only be used through the rcExecMyRule (irule) call i.e., rule execution initiated by clients
and should not be called internally by the server since it interacts with the client through the normal
client/server socket connection. Also, it should never be called through delay since it requires client
interaction.

Example Usage:
myTestRule {
  # Input parameters are:
  # Data object path
  # Optional resource
  # Optional flags in form keyword=value
  # localPath
  # destRescName
  # all - to upload to all resources
  # forceFlag=
  # replNum - the replica number to overwrite
  # numThreads
  # filePath - the physical file path of the uploaded file on the server
  # dataType
  # verifyChksum=
  # Output parameter is:
  # Status
  # Output from running the example is:
# File /tempZone/home/rods/sub1/foo1 is written to the data grid as foo1
msiDataObjPut(*DestFile,*DestResource,"localPath=*LocalFile++++forceFlag=",*Status);
writeLine("stdout","File *LocalFile is written to the data grid as *DestFile");
}

INPUT *DestFile="/tempZone/home/rods/sub1/foo1", *DestResource="demoResc", *LocalFile="foo1"
OUTPUT ruleExecOut
4.29  Core :: Data Object :: msiDataObjPutWithOptions

msiDataObjPutWithOptions ( msParam_t* inpParam1,
msParam_t* inpParam2,
msParam_t* inpParam3,
msParam_t* inpOverwriteParam,
msParam_t* inpAllCopiesParam,
msParam_t* outParam )

Parameters:
[in]  inpParam1 - a DataObjInp_MS_T or a STR_MS_T which would be
taken as dataObj path.
[in]  inpParam2 - Optional - a STR_MS_T which specifies the resource.
[in]  inpParam3 - Optional - a STR_MS_T which specifies the client's local file path.
[in]  inpOverwriteParam - Optional - a STR_MS_T which specifies if the put should
do an overwrite if content already exists in the resource.
    To trigger an overwrite, the "forceFlag" keyword is expected
[in]  inpAllCopiesParam - Optional - a STR_MS_T which specifies if that in case of
    an overwrite, the operation should overwrite all existing copies. Use
    the keyword "all" to overwrite all copies.
[out] outParam - a INT_MS_T containing the status.

Description:
This micro-service puts an object into the data grid by requesting the client to call a rcDataObjPut API as
part of a workflow execution, with options input as separate parameters.

Note:
This micro-service is deprecated, and is replaced with msiDataObjPut.

Example Usage:

myTestRule {
    # Input parameters are:
    #   Data object path
    #   Optional resource
    #   Optional local file path
    #   Optional flag forceFlag
    #   Optional flag all - to overwrite all copies
    # Output parameter is:
    #   Status
    # Output from running the example is:
    # File foo1 is written into the data grid as file /tempZone/home/rods/sub1/foo1
    msiDataObjPutWithOptions(*DestFile,*DestResource,*LocalFile,"forceFlag","",*Status);
    writeLine("stdout","File *LocalFile is written into the data grid as file *DestFile");
}

INPUT *DestFile="/tempZone/home/rods/sub1/foo1", *DestResource="demoResc", *LocalFile="foo1"
OUTPUT ruleExecOut
4.30  Core :: Data Object :: msiDataObjRename

msiDataObjRename (msParam_t * inpParam1,
msParam_t * inpParam2,
msParam_t * inpParam3,
msParam_t * outParam)

Parameters:
[in]  inpParam1 - A DataObjCopyInp_MS_T or STR_MS_T which would be taken
   as the src dataObj path.
[in]  inpParam2 - Optional - A DataObjInp_MS_T which is the destination
   DataObjInp or STR_MS_T which would be the destination object Path.
[in]  inpParam3 - Optional - a INT_MS_T or STR_MS_T which specifies the
   object type. A 0 means data obj and > 0 means collection.
[out] outParam - a INT_MS_T containing the status.

Description:
This micro-service calls rsDataObjRename to rename the input data object or collection to another path. The
destination path name cannot exist before the call. All replicas are changed to the new name.

Example Usage:

myTestRule {
  # Input parameters are:
  # Source data object path
  # Optional destination object path
  # Optional Object type
  # 0 means data object
  # 1 means collection
  # Output parameter is:
  # Status
  # Output from running the example is:
  # The name of /tempZone/home/rods/sub1/foo1 is changed to /tempZone/home/rods/sub1/foo2
  msiDataObjRename(*SourceFile,*NewFilePath,"0",*Status);
  # To change the name of a collection, set the third input parameter to 1
  writeLine("stdout","The name of *SourceFile is changed to *NewFilePath");
}
INPUT *SourceFile="/tempZone/home/rods/sub1/foo1",
*NewFilePath="/tempZone/home/rods/sub1/foo2"
OUTPUT ruleExecOut

93
4.31 Core :: Data Object :: msiDataObjRepl

msiDataObjRepl ( msParam_t * inpParam1,
                 msParam_t * msKeyValStr,
                 msParam_t * outParam )

Parameters:
[in]  inpParam1 - a msParam of type DataObjInp_MS_T or STR_MS_T which
                 would be the obj Path.
[in]  msKeyValStr - Optional - a STR_MS_T. This is the special msKeyValStr format
                 of keyWd1=value1++++keyWd2=value2++++keyWd3=value3...
                 If the keyWd is not specified (without the '=' char), the value is
                 assumed to be the target resource ("destRescName") for
                 backward compatibility. Valid keyWds are:
                 "destRescName" - the target resource to replicate to.
                 "backupRescName" - the target resource to backup the data. If
                 this keyWd is used, the backup mode will
                 be switched on.
                 "rescName" - the resource
                 of the source copy.
                 "updateRepl" - update other replicas with the latest copy.
                 This keyWd has no value. But the '='
                 character is still needed.
                 "replNum" - the replica number to use as source.
                 "numThreads" - the number of threads to use.
                 "all" - replicate to all resources in the resource
                 group. This keyWd has no value.
                 "irodsAdmin" - admin user replicate other users'
                 files. This keyWd has no value.
                 "verifyChksum" - verify the transfer using checksum. This
                 keyWd has no value.
                 "rbudpTransfer" - use RBUDP (datagram) protocol for the
                 data transfer. This keyWd has no value.
                 "rbudpSendRate" - Valid only if "rbudpTransfer" is on. This is
                 the send rate in kbits/sec. The default is 600,000.
                 "rbudpPackSize" - Valid only if "rbudpTransfer" is on. This is
                 the packet size in bytes. The default is 8192.
[out] outParam - a msParam of type INT_MS_T which is the status of the operation.

Description:
This micro-service replicates a file in a collection (it assigns a different replica number to the new copy in
the iCAT Metadata Catalog).

Note:
Can be called by client through irule. In the example, the replica is physically stored in the "testResc"
resource.

Example Usage:

myTestRule {
  # Input parameters are:
  # Data Object path
  # Optional flags in form keyword=value
  #  destRescName - the target resource for the replica
  #  backupRescName - specifies use of the resource for the backup mode
  #  rescName - the resource holding the source data
  #  updateRepl= - specifies all replicas will be updated

# replNum - specifies the replica number to use as the source
# numThreads - specifies the number of threads to use for transmission
# all - specifies to replicate to all resources in a resource group
# irodsAdmin - enables administrator to replicate other users’ files
# verifyChksum - verify the transfer using checksums
# rbudpTransfer - use Reliable Blast UDP for transport
# rbudpSendRate - the transmission rate in kbits/sec, default is 600 kbits/sec
# rbudpPackSize - the packet size in bytes, default is 8192
# Output parameter is:
# Status
# Output from running the example is:
# The file /tempZone/home/rods/sub1/foo3 is replicated onto resource testResc
msiDataObjRepl(*SourceFile,"destRescName=*Resource","*Status);
writeLine("stdout","The file *SourceFile is replicated onto resource *Resource");
}
INPUT *SourceFile="/tempZone/home/rods/sub1/foo3", *Resource="testResc"
OUTPUT ruleExecOut
4.32 Core :: Data Object :: msiDataObjReplWithOptions

msiDataObjReplWithOptions (msParam_t * inpParam1,
                          msParam_t * inpParam2,
                          msParam_t * inpParam3,
                          msParam_t * outParam)

Parameters:
[in] inpParam1 - a msParam of type DataObjInp_MS_T or STR_MS_T which would be the obj Path.
[in] inpParam2 - Optional - a STR_MS_T which specifies the resource.
[in] inpParam3 - Optional - a STR_MS_T which specifies an additional parameter
   - all - update all replicas
   - irodsAdmin - administrator controlled replication
[out] outParam - a INT_MS_T for the status.

Description:
This micro-service is the same as msiDataObjRepl, but with options input as separate parameters.

Note:
This micro-service is deprecated, and is replaced with msiDataObjRepl. Since iRODS version 2.2, msiDataObjRepl can take a parameter with key-values format. Can be called by client through irule.

Example Usage:

myTestRule {
   # This microservice is deprecated
   # Input parameters are:
   # Data object path
   # Optional storage resource
   # Optional parameter, valid flags are:
   # all - to specify replicate to all resources in storage group
   # irodsAdmin - to enable administrator to replicate other users' files
   msiDataObjReplWithOptions(*SourceFile,*Resource,"null",*Status);
   writeLine("stdout","The file *SourceFile is replicated onto resource *Resource");
}
INPUT *SourceFile="/tempZone/home/rods/sub1/foo2", *Resource="testResc"
OUTPUT ruleExecOut
msiDataObjRsync ( msParam_t * inpParam1,
                  msParam_t * inpParam2,
                  msParam_t * inpParam3,
                  msParam_t * inpParam4,
                  msParam_t * outParam )

Parameters:
[in] inpParam1 - A DataObjInp_MS_T or STR_MS_T which would be taken as
                dataObj path.
[in] inpParam2 - Optional - a STR_MS_T which specifies the rsync mode.
                Valid mode is IRODS_TO_IRODS and
                IRODS_TO_COLLECTION.
[in] inpParam3 - Optional - a STR_MS_T which specifies the resource name.
[in] inpParam4 - Optional - a STR_MS_T which specifies the RSYNC_DEST_PATH_KW,
                rsyncDestPath. For IRODS_TO_IRODS, this is
                the target path. For IRODS_TO_COLLECTION, this is the top
                level target collection. e.g., if dataObj (inpParam1) is
                /tempZone/home/rods/foo and the target collection (inpParam4) is
                /tempZone/archive, then the target path is
                /tempZone/archive/home/rods/foo.
[out] outParam - a INT_MS_T containing the status.

Description:
This micro-service synchronizes a data object with the data grid by requesting
the client to call a
rcDataObjRsync API as part of a workflow execution.

Note:
For now, this micro-service should only be used for IRODS_TO_IRODS mode because of the logistic
difficulty with the micro-service getting the checksum values of the local file.

Example Usage:

myTestRule {
  # Input parameters are:
  # Data object path
  # Optional flag for mode
  # IRODS_TO_IRODS
  # IRODS_TO_COLLECTION
  # Optional storage resource
  # Optional target collection
  # Output parameters are:
  # Status
  # Output from running the example is:
  # The file /tempZone/home/rods/sub1/foo2 is synchronized onto the logical data object path
  msiDataObjRsync(*SourceFile,"IRODS_TO_IRODS",*DestResource,*DestPathName,*Status);
  writeLine("stdout","The file *SourceFile is synchronized onto the logical data object path
  *DestPathName");
} INPUT
*SourceFile="/tempZone/home/rods/sub1/foo2",*DestResource="testResc",*DestPathName="/tempZone/h
ome/rods/rules"
OUTPUT ruleExecOut
4.34 Core :: Data Object :: msiDataObjTrim

\[
\text{msiDataObjTrim}( \text{msParam}_t^{*} \text{inpParam1}, \\
\text{msParam}_t^{*} \text{inpParam2}, \\
\text{msParam}_t^{*} \text{inpParam3}, \\
\text{msParam}_t^{*} \text{inpParam4}, \\
\text{msParam}_t^{*} \text{inpParam5}, \\
\text{msParam}_t^{*} \text{outParam})
\]

**Parameters:**

- **[in]** `inpParam1` - A DataObjInp_MS_T or STR_MS_T which would be taken as dataObj path.
- **[in]** `inpParam2` - Optional - a STR_MS_T which specifies the resourceName.
- **[in]** `inpParam3` - Optional - a STR_MS_T which specifies the replNum.
- **[in]** `inpParam4` - Optional - a STR_MS_T which specifies the minimum number of copies to keep.
- **[in]** `inpParam5` - Optional - a STR_MS_T which specifies administrator controlled trimming of replicas irodsAdmin – flag to indicate actions by an administrator
- **[out]** `outParam` - a INT_MS_T containing the status.

**Description:**
This micro-service calls rsDataObjTrim to trim down the number of replicas of a data object.

**Example Usage:**

```plaintext
myTestRule {
    # Input parameters are:
    # Data object path
    # Optional storage resource name
    # Optional replica number
    # Optional number of replicas to keep
    # Optional administrator flag irodsAdmin, to enable administrator to trim replicas
    # Output parameter is:
    # Status
    # Output from running the example is:
    # The replicas of File /tempZone/home/rods/sub1/foo2 are deleted
    msiDataObjTrim(*SourceFile,"null","null","1","null",*Status);
    writeLine("stdout","The replicas of file *SourceFile are deleted");
}
INPUT *SourceFile="/tempZone/home/rods/sub1/foo2"
OUTPUT ruleExecOut
```

98
4.35 Core :: Data Object :: msiDataObjUnlink

msiDataObjUnlink ( msParam_t * inpParam,
                  msParam_t * outParam )

Parameters:
[in] inpParam - a msParam of type DataObjInp_MS_T or STR_MS_T which
would be taken as msKeyValStr. msKeyValStr - This is the special msKeyValStr format of
keyWd1=value1++++keyWd2=value2++++keyWd3=value3... If
the keyWd is not specified (without the '=' char), the value is
assumed to be the path of the data object("objPath") for backward
compatibility. Valid keyWds are:
  "objPath" - the path of the data object to remove.
  "replNum" - the replica number of the copy to remove.
  "forceFlag=" - Remove the data object instead of putting it in trash.
               This keyWd has no value. But the '=' character is
               still needed.
  "irodsAdminRmTrash=" - Admin rm trash. This keyWd has no value.
  "irodsRmTrash=" - rm trash. This keyWd has no value.
[out] outParam - a msParam of type INT_MS_T for the status.

Description:
This micro-service deletes an existing data object.

Note:
Can be called by client through irule. When used with irodsRmTrash, the objPath must specify a file
within the trash.

The msiDataObjUnlink micro-service will not delete a collection.

Example Usage:

myTestRule {
  # Input parameter is:
  # Flags in form keyword=value
  # objPath - the data object path to remove
  # replNum - the replica number to be removed
  # forceFlag= - flag to remove file without transferring to trash
  # irodsAdminRmTrash - flag to allow administrator to remove trash
  # irodsRmTrash - flag for user to remove trash
  # Output parameter is:
  # Status
  # Output from running the example is:
  # Replica number 1 of file /tempZone/home/rods/sub1/foo3 is removed
  msiDataObjUnlink("objPath=*SourceFile++++replNum=1","Status");
  writeLine("stdout","Replica number 1 of file *SourceFile is removed");
}
INPUT *SourceFile="/tempZone/home/rods/sub1/foo3"
OUTPUT ruleExecOut
4.36 Core :: Data Object :: msiGetObjType

**msiGetObjType** (msParam_t * objParam, msParam_t * typeParam)

**Parameters:**
- **[in] objParam** - a msParam of type STR_MS_T, the path of the iRODS object
- **[out] typeParam** - a msParam of type STR_MS_T, output value of the object type

**Description:**
This micro-service gets an object's type from the iCAT to specify whether file, collection resource, or user.

**Note:**
Valid object types are:
- **-d** file
- **-c** collection
- **-r** resource
- **-g** resource group
- **-u** user
- **-m** metadata
- **-t** token

**Example Usage:**

```c
myTestRule {
    # Input parameter is:
    # Object name
    # Output parameter is:
    # Type
    # Output from running the example is:
    # The type of object /tempZone/home/rods/sub1/foo3 is -d
    # The type of object demoResc is -r
    msiGetObjType(*SourceFile,*Type);
    writeLine("stdout","The type of object *SourceFile is *Type");
    msiGetObjType(*Resource,*Type1);
    writeLine("stdout","The type of object *Resource is *Type1");
}

INPUT *SourceFile="/tempZone/home/rods/sub1/foo3", *Resource="demoResc"
OUTPUT ruleExecOut
```
4.37  Core :: Data Object :: msiObjStat

**msiObjStat** (msParam_t * inpParam1, msParam_t * outParam)

**Parameters:**

- **[in]** inpParam1 - A DataObjInp_MS_T or STR_MS_T which would be taken as dataObj path.
- **[out]** outParam - a RodsObjStat_PI structure containing an integer with value COLL_OBJ_T (collection or DATA_OBJ_T (data object).

**Description:**

This micro-service calls rsObjStat to get the stat of an iRODS path as part of a workflow execution.

**Example Usage:**

```c
myTestRule {
  # Input parameter is:
  # Data object path
  # Output parameter is:
  # Type of object is written into a RodsObjStat_PI structure
  msiSplitPath(*SourceFile,*Coll,*File);
  msiObjStat(*SourceFile,*Stat);
  msiObjStat(*Coll,*Stat1);
  writeLine("stdout","Type of object is written into a RodsObjStat_PI structure");
}
```

INPUT *SourceFile="/tempZone/home/rods/sub1/foo3"
OUTPUT ruleExecOut
4.38 Core :: Data Object :: msiPhyPathReg

**msiPhyPathReg**

```
(msParam_t * inpParam1,
msParam_t * inpParam2,
msParam_t * inpParam3,
msParam_t * inpParam4,
msParam_t * outParam)
```

**Parameters:**

- **inpParam1** - A DataObjInp_MS_T or STR_MS_T which would be taken as object path. The path can be a data object or a collection path.
- **inpParam2** - Optional - a STR_MS_T which specifies the dest resourceName.
- **inpParam3** - Optional - a STR_MS_T which specifies the physical path to be registered.
- **inpParam4** - Optional - a STR_MS_T which specifies whether the path to be registered is a directory. A keyword string "collection" indicates the path is a directory. A "null" string indicates the path is a file. A keyword string "mountPoint" (MOUNT_POINT_STR) means mount the file directory given in inpParam3. A keyword string "linkPoint" (LINK_POINT_STR) means soft link the collection given in inpParam3.

- **outParam** - a INT_MS_T containing the status.

**Description:**
This micro-service calls rsPhyPathReg to register a physical path with the iCAT.

**Note:**
The data object path in iRODS must be created before the registration is done.

**Example Usage:**

```rsh
myTestRule {
    # Input parameters are:
    # Data object path
    # Optional destination resource
    # Optional physical path to register
    # Optional flag for type of
    #   collection - specifies the path is a directory
    #   null - specifies the path is a file
    #   mountPoint - specifies to mount the physical path
    #   linkPoint - specifies soft link the physical path
    # Output parameter is:
    # Status
    # Output from running the example is:
    # The local collection /home/reagan/irods-scripts/ruletest is mounted under the logical collection
    /tempZone/home/rods/irods-rules
    msiPhyPathReg(*DestCollection,*Resource,*SourceDirectory,"mountPoint",*Stat);
    writeLine("stdout","The local collection *SourceDirectory is mounted under the logical collection
    *DestCollection");
}
```

INPUT *DestCollection="/tempZone/home/rods/irods-rules", *SourceDirectory="/home/reagan/irods-scripts/ruletest", *Resource="demoResc"

OUTPUT ruleExecOut
4.39 Core :: Data Object :: msiSetReplComment

msiSetReplComment ( msParam_t * inpParam1,
     msParam_t * inpParam2,
     msParam_t * inpParam3,
     msParam_t * inpParam4 )

Parameters:
[in] inpParam1 - a INT with the id of the object (can be null if unknown, the next
param will then be used)
[in] inpParam2 - a msParam of type DataObjInp_MS_T or a STR_MS_T which
would be taken as dataObj path
[in] inpParam3 - a INT which gives the replica number
[in] inpParam4 - a STR_MS_T containing the comment

Description:
This micro-service sets the data_comments attribute of a data object.

Note:
Can be called by client through irule

Example Usage:

myTestRule {    
    # Input parameters are:  
    # Object ID if known  
    # Data object path  
    # Replica number  
    # Comment to be added  
    # Output parameter is:  
    # Status  
    # Output from running the example is:  
    # The comment added to file /tempZone/home/rods/sub1/foo3 is "New comment"  
    # The comment retrieved from iCAT is "New comment"
    msiSetReplComment("null",*SourceFile,0,*Comment);
    writeLine("stdout","The comment added to file *SourceFile is *Comment");
    msiSplitPath(*SourceFile,*Coll,*File);
    msiMakeGenQuery("DATA_COMMENTS","DATA_NAME = '*File' AND COLL_NAME = '*Coll'",*GenQInp);
    msiExecGenQuery(*GenQInp,*GenQOut);
    foreach(*GenQOut) {
        msiGetValByKey(*GenQOut,"DATA_COMMENTS",*com);
        writeLine("stdout","The comment retrieved from iCAT is *com");
    }
    }
INPUT *SourceFile="/tempZone/home/rods/sub1/foo3", *Comment="New comment"
OUTPUT ruleExecOut
4.40  Core :: Helper :: msiAddKeyValToMspStr

msiAddKeyValToMspStr      (   msParam_t *   keyStr,
                                msParam_t *   valStr,
                                msParam_t *   msKeyValStr   )

Parameters:
  [in]   keyStr    - a STR_MS_T key to be added to msKeyValStr.
  [in]   valStr    - a STR_MS_T value to be added to msKeyValStr.
  [in]   msKeyValStr    - a msKeyValStr to hold the new keyVal pair.

Description:
Adds a key and value to existing msKeyValStr which is a special kind of STR_MS_T that has the format -
keyWd1=value1+++keyWd2=value2+++keyWd3=value3...

Note:
none

Example Usage:

myTestRule {  
  # Input parameters are:
  #  Attribute name   
  #  Attribute value
  # Output from running the example is:
  #  The string now contains
  #  destRescName=demoResc
  msiAddKeyValToMspStr(*AttrName,*AttrValue,*KeyValStr); 
  writeLine("stdout","The string now contains"); 
  writeLine("stdout","*KeyValStr"); 
  }
INPUT *AttrName="destRescName", *AttrValue="demoResc"
OUTPUT ruleExecOut
4.41 Core :: Helper :: msiDoSomething

msiDoSomething ( msParam_t *inParam, msParam_t *outParam )

Parameters:
[in] inParam - Any type. A STR_MS_T can be used to pass multiple parameters in the format
keyWd1=value1++++keyWd2=value2++++keyWd3=value3...
[out] outParam - A KeyValPair_MS_T (by default) containing the keyword-value pairs.

Description:
This empty microservice is to be filled with your own code. It can serve as a platform for quickly testing
server API functions and related code. Input and output parameters can be of any type, depending on how
they are parsed and set up in msiDoSomething. A quick and dirty way to examine critical variables without
firing up gdb is to create a keyValPair_t*, dump data in it throughout the code with addKeyVal(), return it
through outParam, and follow with writeKeyValPairs.

Note:
none

Example Usage:

myTestRule {
    # Placeholder for creating a rule for a new microservice
    msiDoSomething("", *keyValOut);
    writeKeyValPairs("stdout", *keyValOut, " : ");
}
INPUT null
OUTPUT ruleExecOut
4.42 Core :: Helper :: msiGetSessionVarValue

msiGetSessionVarValue ( msParam_t * inpVar,
                         msParam_t * outputMode )

Parameters:
[in] inpVar - A STR_MS_T which specifies the name of the session variable to output. The input session variable should NOT start with the "$" character. An input value of "all" means output all valid session variables.

[in] outputMode - A STR_MS_T which specifies the output mode. Valid modes are "server" - log the output to the server log, "client" - send the output to the client in rError, "all" - send to both client and server

Description:
Gets the value of a session variable from the rei structure in memory

Note:
none

Example Usage:

myTestRule {
    # Input parameters are:
    # Session variable
    # Session variable without the $ sign
    # all - output all of the defined variables
    # Output mode flag:
    # server - log the output to the server log
    # client - send the output to the client in rError
    # all - send the output to both client and server
    # Output from running the example is:
    # Variables are written to the log file
    # Output in irods/server/log/rodsLog.2011.6.1 log file is:
    # msiGetSessionVarValue: userNameClient=rods
        msiGetSessionVarValue(*A,"server");
        writeLine("stdout","Variables are written to the log file");
    }
INPUT *A="userNameClient"
OUTPUT ruleExecOut
### Core :: Helper :: msiGetStderrInExecCmdOut

```c
msiGetStderrInExecCmdOut (msParam_t * inpExecCmdOut, msParam_t * outStr)
```

**Parameters:**

- **[in]** `inpExecCmdOut` - a STR_MS_T which specifies the ExecCmdOut.
- **[out]** `outStr` - a STR_MS_T to hold the retrieved stderr buffer.

**Description:**

Gets stderr buffer from ExecCmdOut into buffer.

**Note:**

none

**Example Usage:**

```c
myTestRule {
    # Only executables stored within irods/server/bin/cmd can be run
    # Input parameter is:
    #  Output buffer from the exec command which holds the status, output, and error messages
    # Output parameter is:
    #  Buffer to hold the retrieved error message
    # Output from running the example is:
    #  Error message is
    msiExecCmd(*Cmd,*ARG,","","",*HELLO_OUT);
    msiGetStderrInExecCmdOut(*HELLO_OUT,*ErrorOut);
    writeLine("stdout","Error message is *ErrorOut");
}
INPUT *Cmd="hello", *ARG="iRODS"
OUTPUT ruleExecOut
```
4.44 Core :: Helper :: msiGetStdoutInExecCmdOut

msiGetStdoutInExecCmdOut (msParam_t * inpExecCmdOut, msParam_t * outStr)

Parameters:
[in] inpExecCmdOut - a STR_MS_T which specifies the ExecCmdOut.
[out] outStr - a STR_MS_T to hold the retrieved stdout buffer.

Description:
Gets stdout buffer from ExecCmdOut into string buffer.

Note:
none

Example Usage:

myTestRule {
    # Input parameter is:
    # Buffer holding the status, output and error messages from the command execution
    # Output parameter is:
    # Buffer holding the output message
    # Output from executing the command is
    # Output message is Hello World iRODS from irods
    msiExecCmd("hello",*ARG,"","","","",*HELLO_OUT);

    # *HELLO_OUT holds the status, output and error messages
    msiGetStdoutInExecCmdOut(*HELLO_OUT,*Out);
    writeLine("stdout","Output message is *Out");
}

INPUT *ARG="iRODS"
OUTPUT ruleExecOut
4.45 Core :: Helper :: msiSplitPath

```c
msiSplitPath ( msParam_t * inpPath,
               msParam_t * outParentColl,
               msParam_t * outChildName )
```

**Parameters:**
- [in] `inpPath` - a STR_MS_T which specifies the pathname to split.
- [out] `outParentColl` - a STR_MS_T to hold the returned parent path.
- [out] `outChildName` - a STR_MS_T to hold the returned child value.

**Description:**
Splits a pathname into parent and child values.

**Note:**

**Example Usage:**

```c
myTestRule {  
  # Input parameter is:
  #  Data object path
  #  Collection name
  #  File name
  # Output from running the example is:
  #  Object is /tempZone/home/rods/sub1/foo1
  #  Collection is /tempZone/home/rods/sub1 and file is foo1
  writeLine("stdout","Object is *dataObject");
  msiSplitPath(*dataObject,*Coll,*File);
  writeLine("stdout","Collection is *Coll and file is *File");
}
```

INPUT *dataObject="/tempZone/home/rods/sub1/foo1"  
OUTPUT ruleExecOut
### 4.46 Core :: Helper :: msiWriteRodsLog

```c
msiWriteRodsLog ( msParam_t * inpParam1,
                   msParam_t * outParam )
```

**Parameters:**

- **[in] inpParam1** - A STR_MS_T which specifies the message to log.
- **[out] outParam** - An INT_MS_T containing the status.

**Description:**

Writes a message into iRODS/server/log/rodsLog.

**Note:**

This call should only be used through the rcExecMyRule (irule) call i.e., rule execution initiated by clients and should not be called internally by the server since it interacts with the client through the normal client/server socket connection.

**Example Usage:**

```c
def myTestRule {
    # Input parameter is:
    # Message to send to iRODS server log file
    # Output parameter is:
    # Status
    # Output from running the example is:
    # Message is Test message for irods/server/log/rodsLog
    # Output written to log file is:
    # msiWriteRodsLog message: Test message for irods/server/log/rodsLog
    writeLine("stdout","Message is *Message");
    msiWriteRodsLog(*Message,*Status);
}
```

**INPUT** *Message=“Test message for irods/server/log/rodsLog“

**OUTPUT** ruleExecOut
4.47 Core :: Helper :: msiExit

msiExit ( msParam_t * inpParam1,
           msParam_t * inpParam2 )

Parameters:
[in] inpParam1 - A STR_MS_T which specifies the status error to add to the error stack.
[in] inpParam2 - A STR_MS_T which specifies the message to add to the error stack.

Description:
Add a user message to the error stack.

Note:
This call should only be used through the rcExecMyRule (irule) call i.e., rule execution initiated by clients and should not be called internally by the server since it interacts with the client through the normal client/server socket connection.

Example Usage:

myTestRule {
  # Input parameters are:
  # Status error to add to the error stack
  # Message to add to the error stack
  # Output from running the example is:
  # Error number 200 and message Test Error
  writeLine("stdout","Error number *Error and message *Message");
  msiExit(*Error,*Message);
}
INPUT *Error="200", *Message="Test Error"
OUTPUT ruleExecOut
4.48  Core :: Proxy Command :: msiExecCmd

```c
msiExecCmd ( msParam_t * inpParam1,
              msParam_t * inpParam2,
              msParam_t * inpParam3,
              msParam_t * inpParam4,
              msParam_t * inpParam5,
              msParam_t * outParam )
```

**Parameters:**

- **[in]** `inpParam1` - a ExecCmd_MS_T or a STR_MS_T which specifies the command (cmd) to execute.
- **[in]** `inpParam2` - Optional - a STR_MS_T which specifies the argv (cmdArgv) of the command.
- **[in]** `inpParam3` - Optional - a STR_MS_T which specifies the host address (execAddr) to execute the command.
- **[in]** `inpParam4` - Optional - a STR_MS_T which specifies an iRODS file path (hintPath). The command will be executed on the host where this file is stored.
- **[in]** `inpParam5` - Optional - A INT_MS_T or a STR_MS_T. If it is greater than zero, the resolved physical path from the logical hintPath (inpParam4) will be used as the first argument in the command.

- **[out]** `outParam` - a ExecCmdOut_MS_T containing the status of the command execution and the stdout/stderr output.

**Description:**

This micro-service requests the client to call a rcExecCmd API to fork and execute a command that resides in the iRODS/server/bin/cmd directory.

**Note:**

This call does not require client interaction, which means it can be used through rcExecMyRule (irule) or internally by the server. Only commands that are in the irods/server/bin/cmd directory can be run.

**Example Usage:**

```c
myTestRule {
    # Input parameters are:
    # Command to be executed located in directory irods/server/bin/cmd
    # Optional command argument
    # Optional host address for command execution
    # Optional hint for remote data object path, command is executed on host where the file is stored
    # Optional flag. If > 0, use the resolved physical data object path as first argument
    # Output parameter is:
    # Structure holding status, stdout, and stderr from command execution
    # Output from running the example is:
    # Command result is
    # Hello world written from irods
    msiExecCmd(*Cmd,*Arg,"null","null","null",*Result);
    msiGetStdoutInExecCmdOut(*Result,*Out);
    writeLine("stdout","Command result is");
    writeLine("stdout","*Out");
}
```

```c
INPUT *Cmd="hello", *Arg="written"
```

```c
OUTPUT ruleExecOut
```
4.49 Core :: Rule Engine :: msiAdmAddAppRuleStruct

**msiAdmAddAppRuleStruct**

```
msParam_t * reFilesParam,
msParam_t * dvmFilesParam,
msParam_t * fnmFilesParam
```

**Parameters:**

- **reFilesParam** - a msParam of type STR_MS_T, which is an application Rules file name without the .re extension.
- **dvmFilesParam** - a msParam of type STR_MS_T, which is a variable name mapping file without the .dvm extension.
- **fnmFilesParam** - a msParam of type STR_MS_T, which is an application micro-service mapping file name without the .fnm extension.

**Description:**

This is a micro-service that reads the specified files in the configuration directory 'server/config/reConfigs' and adds them to the in-memory structures being used by the Rule Engine. These rules are loaded before the rules from the "core.re" file, and hence can be used to override the core rules from the "core.re" file (i.e., it adds application level rules and DVM and FNM mappings to the rule engine).

**Note:**

This micro-service requires iRODS administration privileges and adds the given rules (re) file, S-variable mapping (dvm) and micro-service logical name mapping (fnm) files to the working memory of the rule engine. Any subsequent rule or micro-services will also use the newly prepended rules and mappings.

Rules are maintained in three locations:

- A "core.re" file that is the current set of rules.
- An In-Memory Rule Base (App Rule Struct) that holds the rules used during a session. This has three parts: rules from the "core.re" file, application rules loaded by **msiAdmAddAppRuleStruct**, and rules executed from the irule command.
- An iCAT database table that manages persistent versions of rules.

**Example Usage:**

```
myTestRule {
  # Examples are in irods/server/config/reConfigs
  # Input parameters are:
  # Rule file without the .re extension
  # Session variable file name mapping file without the .dvm extension
  # Application micro-service mapping file without the .fnm extension
  # Output from running the example is:
  # List of the rules in the In-memory Rule Base
  msiAdmAddAppRuleStruct("*File","","");
  msiAdmShowIRB();
}
```

**INPUT** *File="core3"
**OUTPUT** ruleExecOut
4.50 Core :: Rule Engine :: msiAdmAppendToTopOfCoreIRB

msiAdmAppendToTopOfCoreIRB ( msParam_t * newFileNameParam )

Parameters:
[in] newFileNameParam - is a msParam of type STR_MS_T, which is a *.irb file name that is prepended to the top of the core.irb file.

Description:
This is a micro-service that changes the core.irb file currently in the configuration directory "server/config/reConfigs" by prepending the given rules file to it. When the server is started next time, then the new core file will be used by the rule engine if the old rule engine is enabled.

Note:
This micro-service requires iRODS administration privileges. This is used with the old rule engine. This micro-service expects the prepended file to be of the form *.irb and to be located in the configuration directory.

If the new rule engine is being used, changes should be made to the "core.re" file instead.

Whenever a micro-service has CoreIRB in the name, the micro-service manipulates the original core.irb file instead of the in-memory rule base.

Example Usage:

myTestRule {
# Input parameter is:
# Core file name that will be prepended including the .irb extension
  msiAdmAppendToTopOfCoreIRB(*A);
}
INPUT *A="core3.irb"
OUTPUT ruleExecOut
4.51 Core :: Rule Engine :: msiAdmAppendToTopOfCoreRE

msiAdmAppendToTopOfCoreRE (msParam_t * newFileNameParam)

Parameters:
[in] newFileNameParam - is a msParam of type STR_MS_T, which is a *.re file that is prepended to the top of the "core.re" file.

Description:
This is a micro-service that changes the "core.re" file currently in the configuration directory "server/config/reConfigs" by prepending the given rules file to it. When the server is started next time, then the new core file will be used by the rule engine.

Note:
This micro-service requires iRODS administration privileges. This micro-service expects the prepended file to be of the form *.re and to be located in the configuration directory, "server/config/reConfigs".

Rules are maintained in three locations:
• A "core.re" file that is the current set of rules.
• An In-Memory Rule Base (App Rule Struct) that holds the rules used during a session. This has three parts: rules from the "core.re" file, application rules loaded by msiAdmAddAppRuleStruct, and rules executed from the irule command.
• An iCAT Database table that manages persistent versions of rules.

Example Usage:

myTestRule {
# Input parameter is:
# Core file name that will be prepended excluding the .re extension
# Output from running the example is:
# Listing of the core.re file
  msiAdmAppendToTopOfCoreRE(*A);
  msiAdmShowCoreRE();
}
INPUT *A="core3"
OUTPUT ruleExecOut
4.52  Core :: Rule Engine :: msiAdmChangeCoreIRB

**msiAdmChangeCoreIRB**

Parameters:

- **newFileNameParam** - is a msParam of type STR_MS_T, which is a new core file name with the .irb extension.

Description:

This micro-service copies the specified file in the configuration directory "server/config/reConfigs" onto the core.irb file in the same directory.

Note:

This micro-service expects the alternate file to be of the form *.irb and to be located in the configuration directory. This should be used only if the old rule engine is used.

This micro-service requires iRODS administration privilege.

This micro-service changes the core.irb file currently in the configuration directory. If the new rule engine is being used, the changes should be made to the "core.re" file instead.

Example Usage:

```plaintext
myTestRule {
  # Input parameter is:
  # Name of file to replace the core.irb file
  msiAdmChangeCoreIRB(*A);
}

INPUT *A="core-orig.irb"
OUTPUT ruleExecOut
```
### 4.53 Core :: Rule Engine :: msiAdmChangeCoreRE

**msiAdmChangeCoreRE**

```
( msParam_t * newFileNameParam )
```

**Parameters:**

- **newFileNameParam**
  - is a msParam of type STR_MS_T, which is a new core file name without the .re extension.

**Description:**

This micro-service copies the specified file in the configuration directory "server/config/reConfigs" onto the "core.re" file in the same directory.

**Note:**

- This micro-service expects the alternate file to be of the form *.re and to be located in the configuration directory.
- This micro-service requires iRODS administration privilege.
- This micro-service changes the "core.re" file currently in the configuration directory. It can be invoked through an irule. When the server is re-started, the new core file will be used by the rule engine.

Rules are maintained in three locations:

- A "core.re" file that is the current set of rules.
- An In-Memory Rule Base (App Rule Struct) that holds the rules used during a session. This has three parts: rules from the "core.re" file, application rules loaded by `msiAdmAddAppRuleStruct`, and rules executed from the irule command.
- An iCAT Database table that manages persistent versions of rules.

**Example Usage:**

```c
myTestRule {
    # Input parameter is:
    # Name of file to replace the core.re file without the .re extension
    # Output
    # Listing of the core.re file
    msiAdmChangeCoreRE(*A);
    msiAdmShowCoreRE();
}
```

```
INPUT *A="core-new-rule"
OUTPUT ruleExecOut
```
4.54  Core :: Rule Engine :: msiAdmClearAppRuleStruct

int msiAdmClearAppRuleStruct ( )

Parameters:
N/A.

Description:
This is a micro-service that clears the application level Rules and DVM and FNM mappings that were loaded into the rule engine's working memory.

Note:
This micro-service needs iRODS administration privileges to perform this function. Clears the application structures in the working memory of the rule engine holding the rules, $-variable mappings and micro-service name mappings.

Rules are maintained in three locations:
- A "core.re file" that is the current set of rules
- An In-Memory Rule Base (App Rule Struct) that holds the rules used during a session. This has three parts: rules from the "core.re" file, application rules loaded by msiAdmAddAppRuleStruct, and rules executed from the irule command
- An iCAT database table that manages persistent versions of rules

Example Usage:

myTestRule {  
  # No Input parameter  
  # Output from running the example:  
  # List of rules after adding rule and after clearing rules  
  msiAdmAddAppRuleStruct(*A,"","","");  
  msiAdmShowIRB();  
  msiAdmClearAppRuleStruct();  
  msiAdmShowIRB();  
}  
INPUT *A="nara"  
OUTPUT ruleExecOut
4.55 Core :: Rule Engine :: msiAdmInsertRulesFromStructIntoDB

`msiAdmInsertRulesFromStructIntoDB (msParam_t * inIrbBaseNameParam, msParam_t * inCoreRuleStruct)`

**Parameters:**

[in] `inIrbBaseNameParam` - a `msParam` of type `STR_MS_T`, which is name of the base that is being added.

[in] `inCoreRuleStruct` - a `msParam` of type `RuleStruct_MS_T` containing the rules.

**Description:**

This is a micro-service that reads the contents of a rule structure and writes them as a new rule base into the core rule tables of the iCAT. It also maintains versioning of the rule base in the iCAT by giving an older version number to the existing base set of rules.

**Note:**

This micro-service requires iRODS administration privileges.

Rules are maintained in three locations:

- A "core.re" file that is the current set of rules.
- An In-Memory Rule Base (App Rule Struct) that holds the rules used during a session. This has three parts: rules from the "core.re" file, application rules loaded by `msiAdmAddAppRuleStruct`, and rules executed from the `irule` command.
- An iCAT database table that manages persistent versions of rules.

**Example Usage:**

```plaintext
MyTestRule {
  # Input parameters are:
  # Name of base that is being added
  # Buffer containing rules
  # Output from running the example:
  # List of rules in the In-memory Rule Base
  msiAdmReadRulesFromFileIntoStruct(*FileName,*Struct);
  msiAdmInsertRulesFromStructIntoDB(*RuleBase,*Struct);
  msiAdmRetrieveRulesFromDBIntoStruct(*RuleBase,"0",*Struct1);
  msiAdmWriteRulesFromStructIntoFile(*FileName1,*Struct1);
  msiAdmAddAppRuleStruct(*FileName1,"","");
  msiAdmShowIRB();
}

INPUT *RuleBase="RajaBase",*FileName="raja",*FileName1="raja1"
OUTPUT ruleExecOut
```
Core :: Rule Engine :: msiGetRulesFromDBIntoStruct

**msiGetRulesFromDBIntoStruct**

```
msParam_t * inIrbBaseNameParam,
msParam_t * inVersionParam,
msParam_t * outCoreRuleStruct
```

**Parameters:**

- **inIrbBaseNameParam** - a msParam of type STR_MS_T, which is the name of the base that is being queried.
- **inVersionParam** - a msParam of type STR_MS_T, which is the version string of the base being queried (use 0 for current version)
- **outCoreRuleStruct** - a msParam of type RuleStruct_MS_T (can be NULL in which case it is allocated)

**Description:**

This is a microservice that queries the iCAT for rules with a given base name and version number and populates a rule structure with those rules.

**Note:**

This microservice requires iRODS administration privileges. It queries rules from the iCAT rule base.

**Example Usage:**

```
MyTestRule {
    # Input parameters are:
    # Name of rule base
    # Version of the rule base
    # Output parameter is:
    # Buffer to hold rules
    # Output from running the example is a list of the rules
    msiAdmReadRulesFromFileIntoStruct(*FileName,*Struct);
    msiAdmInsertRulesFromStructIntoDB(*RuleBase,*Struct);
    msiGetRulesFromDBIntoStruct(*RuleBase,"0",*Struct1);
    msiAdmWriteRulesFromStructToFile(*FileName1,*Struct1);
    msiAdmAddAppRuleStruct(*FileName1,"","");
    msiAdmShowIRB("null");
}
```

INPUT *RuleBase="RajaBase", *FileName="raja", *FileName1="raja1"
OUTPUT ruleExecOut
4.57 Core :: Rule Engine :: msiAdmReadRulesFromFileIntoStruct

**msiAdmReadRulesFromFileIntoStruct** (msParam_t * inIrbFileNameParam, msParam_t * outCoreRuleStruct)

**Parameters:**
- [in] inIrbFileNameParam - a msParam of type STR_MS_T, a Rules file in .re format, either in "server/config/reConfigs/" and without the .re extension, or a full file path in another directory on the server.
- [out] outCoreRuleStruct - of type RuleStruct_MS_T (can be NULL in which case it is allocated)

**Description:**
This is a micro-service that reads the given file in the configuration directory 'server/config/reConfigs' or any file in the server local file system and reads them into a rule structure.

**Note:**
This micro-service requires iRODS administration privileges.

Adds the given rules from a ".re" file to a given rule structure.

Rules are maintained in three locations:
- A "core.re" file that is the current set of rules.
- An In-Memory Rule Base (App Rule Struct) that holds the rules used during a session. This has three parts: rules from the "core.re" file, application rules loaded by **msiAdmAddAppRuleStruct**, and rules executed from the irule command.
- An iCAT database table that manages persistent versions of rules.

**Example Usage:**

```
MyTestRule {
    # Input Parameter is:
    # File containing rules
    # Output parameter is:
    # Buffer holding rule struct
    msiAdmReadRulesFromFileIntoStruct(*FileName,*Struct);
    msiAdmInsertRulesFromStructIntoDB(*RuleBase,*Struct);
    msiAdmRetrieveRulesFromDBIntoStruct(*RuleBase,"0",*Struct1);
    msiAdmWriteRulesFromStructIntoFile(*FileName1,*Struct1);
    msiAdmAddAppRuleStruct(*FileName1,"","";)
    msiAdmShowIRB();
}
```

INPUT *RuleBase="RajaBase",*FileName="raja",*FileName1="raja1"
OUTPUT ruleExecOut
4.58 Core :: Rule Engine :: msiAdmShowCoreRE

msiAdmShowCoreRE ( )

Parameters:
none

Description:
This is a micro-service that prints the "core.re" file currently in the configuration directory "server/config/reConfigs".

Note:
Rules are maintained in three locations:
- A "core.re" file that is the current set of rules.
- An In-Memory Rule Base (App Rule Struct) that holds the rules used during a session. This has three parts: rules from the "core.re" file, application rules loaded by msiAdmAddAppRuleStruct, and rules executed from the irule command.
- An iCAT database table that manages persistent versions of rules.

Example Usage:

myTestRule {
  # Input parameter is:
  # none
  # Output from running the example is:
  # Listing of the core.re file
  msiAdmShowCoreRE();
}
INPUT null
OUTPUT ruleExecOut
4.59  Core :: Rule Engine :: msiAdmShowDVM

msiAdmShowDVM ( msParam_t * bufParam )

Parameters:
[in] bufParam - is a msParam (not used for anything, a dummy parameter)

Description:
This is a micro-service that reads the data-value-mapping data structure in the Rule Engine and pretty-prints that structure to the stdout buffer.

Note:
This micro-service uses a dummy parameter. Lists the currently loaded dollar variable mappings from the rule engine memory. The list is written to stdout in ruleExecOut.

Example Usage:

myTestRule {
# Dummy input argument
# Output from running the example:
# List of Session variable mappings from the rule engine memory
  msiAdmShowDVM(*A);
}
INPUT *A="null"
OUTPUT ruleExecOut
4.60  Core :: Rule Engine :: msiAdmShowFNM

msiAdmShowFNM  (  msParam_t *  bufParam  )

Parameters:
[in]  bufParam  - is a msParam (not used for anything, a dummy parameter)

Description:
This is a micro-service that reads the function-name-mapping data structure in the rule engine and pretty-prints that structure to the stdout buffer.

Note:
This micro-service has a dummy parameter.
This micro-service lists the currently loaded micro-services and action name mappings from the rule engine memory. The list is written to stdout in ruleExecOut.

Example Usage:

myTestRule {
  # Dummy input parameter
  # Output from running the example is a list of the microservice and action name mappings from the rule engine memory
  msiAdmShowFNM(*A);
}
INPUT *A="null"
OUTPUT ruleExecOut
4.61 Core :: Rule Engine :: msiAdmShowIRB

msiAdmShowIRB ( )

Parameters:
None

Description:
This is a micro-service that reads the data structure in the rule engine, which holds the current set of Rules, and pretty-prints that structure to the stdout buffer.

Note:
The IRB term refers to the In-memory Rule Base, to differentiate from the "core.re" file that is read each time a new session is started.

Rules are maintained in three locations:
• A "core.re" file that is the current set of rules
• An In-Memory Rule Base (App Rule Struct) that holds the rules used during a session. This has three parts: rules from the "core.re" file, application rules loaded by msiAdmAddAppRuleStruct, and rules executed from the irule command
• An iCAT database table that manages persistent versions of rules

Example Usage:

myTestRule {
   # Dummy input parameter
   # Output from running the example is:
   # List of rules from the rule engine memory
   msiAdmShowIRB();
}
INPUT *B="null"
OUTPUT ruleExecOut
4.62 Core :: Rule Engine :: msiAdmWriteRulesFromStructIntoFile

**msiAdmWriteRulesFromStructIntoFile** (msParam_t * inIrbFileNameParam, msParam_t * inCoreRuleStruct)

**Parameters:**
- **inIrbFileNameParam** - A msParam of type STR_MS_T, which is either a base-name in which case the file will be written into the "server/config/reConfigs/" directory with a .re extension, or a full file path in another directory on the server.
- **inCoreRuleStruct** - A msParam of type RuleStruct_MS_T

**Description:**
This is a micro-service that writes into a given file the contents of a given rule structure. The file can be in "server/config/reConfigs/" or any path on the server local file system.

**Note:**
This micro-service requires iRODS administration privileges.

Rules are maintained in three locations:
- A core.re file that is the current set of rules
- An In-Memory Rule Base (App Rule Struct) that holds the rules used during a session. This has three parts: rules from the core.re file, application rules loaded by msiAdmAddAppRuleStruct, and rules executed from the irule command
- An iCAT database table that manages persistent versions of rules

**Example Usage:**

```
MyTestRule {
    # Input parameters are:
    # File name for writing rules
    # Rule structure that will be written
    # Output lists the rules in the in-memory rule structure
    msiAdmReadRulesFromFileIntoStruct(*FileName,*Struct);
    msiAdmInsertRulesFromStructIntoDB(*RuleBase,*Struct);
    msiAdmRetrieveRulesFromDBIntoStruct(*RuleBase,"0",*Struct1);
    msiAdmWriteRulesFromStructIntoFile(*FileName1,*Struct1);
    msiAdmAddAppRuleStruct(*FileName1,"",""");
    msiAdmShowIRB();
}
```

**INPUT** *RuleBase="RajaBase", *FileName="raja", *FileName1="raja1"
**OUTPUT** ruleExecOut
4.63 Core :: Rule Engine :: msiAdmRetrieveRulesFromDBIntoStruct

**msiAdmRetrieveRulesFromDBIntoStruct**  
(msParam_t * inIrbBaseNameParam,  
msParam_t * inVersionParam,  
msParam_t * outCoreRuleStruct)

**Parameters:**

- **[in] inIrbBaseNameParam**  - a msParam of type STR_MS_T, which is the name of the base that is being queried.
- **[in] inVersionParam**  - a msParam of type STR_MS_T, which is the version string of the base being queried (use 0 for current version)
- **[out] outCoreRuleStruct**  - a msParam of type RuleStruct_MS_T (can be NULL in which case it is allocated)

**Description:**

This is a micro-service that queries the iCAT for rules with a given base name and version number and populates a rule structure with those rules.

**Note:**

This micro-service requires iRODS administration privileges. Extracts rules from the iCAT rule database.

Rules are maintained in three locations:
- A core.re file that is the current set of rules
- An In-Memory Rule Base (App Rule Struct) that holds the rules used during a session. This has three parts: rules from the core.re file, application rules loaded by msiAdmAddAppRuleStruct, and rules executed from the irule command
- An iCAT database table that manages persistent versions of rules

**Example Usage:**

```
MyTestRule { 
# Input parameters are:  
# Name of rule base  
# Version of the rule base  
# Output parameter is:  
# Buffer to hold rules  
# Output from running the example is a list of the rules  
msiAdmReadRulesFromFileIntoStruct(*FileName,*Struct);  
msiAdmInsertRulesFromStructIntoDB(*RuleBase,*Struct);  
msiAdmRetrieveRulesFromDBIntoStruct(*RuleBase,"0",*Struct1);  
msiAdmWriteRulesFromStructIntoFile(*FileName1,*Struct1);  
msiAdmAddAppRuleStruct(*FileName1,"",""};  
msiAdmShowIRB();  
}
```

**INPUT** *RuleBase*="RajaBase", *FileName="raja", *FileName1="raja1"

**OUTPUT** ruleExecOut
4.64 Core :: String Manipulation :: msiStrlen

msiStrlen (msParam_t * stringIn,
            msParam_t * lengthOut)

Parameters:
[in] stringIn - a STR_MS_T which specifies the input string.
[out] lengthOut - a STR_MS_T to hold the returned string length.

Description:
Returns the length of a given string.

Note:
none

Example Usage:

myTestRule {
  # Input parameter is:
  # String
  # Output parameter is:
  # Length of string
  # Output from running the example is:
  # The String: /tempZone/home/rods/sub1/foo1 has length 29
  msiStrlen(*StringIn,*Length);
  writeLine("stdout","The string: *StringIn has length *Length");
}
INPUT *StringIn="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
4.65  Core :: String Manipulation :: msiStrchop

msiStrchop (  msParam_t *  stringIn,
               msParam_t *  stringOut )

Parameters:
[in]  stringIn   - a STR_MS_T which specifies the input string.
[out] stringOut  - a STR_MS_T to hold the string without the last char.

Description:
Removes the last character of a given string.

Note:
none

Example Usage:

myTestRule {
    # Input parameter is:
    # String
    # Output parameter is:
    # String without the last character
    # Output from running the example is:
    # The input string is: /tempZone/home/rods/sub1/foo1/
    # The output string is: /tempZone/home/rods/sub1/foo1
    msiStrchop(*StringIn,*StringOut);
    writeLine("stdout","The input string is: *StringIn");
    writeLine("stdout","The output string is: *StringOut");
}
INPUT *StringIn="/tempZone/home/rods/sub1/foo1/"
OUTPUT ruleExecOut
4.66  Core :: String Manipulation :: msiSubstr

msiSubstr ( msParam_t * stringIn,
            msParam_t * offset,
            msParam_t * length,
            msParam_t * stringOut )

Parameters:

[in]  stringIn    - a STR_MS_T which specifies the input string.
[in]  offset      - a STR_MS_T which specifies the position of the beginning of the
                    substring (0 is first character). If negative, then offset specifies
                    the position from the end of the string (-1 is the last character).
[in]  length      - a STR_MS_T which specifies the length of substring to return. If
                    length is not specified, too large, negative, or "null", then return
                    the substring from the offset to the end of stringIn.
[out] stringOut   - a STR_MS_T to hold the resulting substring.

Description:
Returns a substring of the given string.

Note:
none

Example Usage:

myTestRule {
    # Input parameters are:
    # String
    # Offset from start counting from 0. If negative, count from end
    # Length of the substring
    # Output parameter is:
    # Substring
    # Output from running the example is:
    # The input string is: /tempZone/home/rods/sub1/foo1/
    # The offset is 10 and the length is 4
    # The output string is: home
    msiSubstr(*StringIn,*Offset,*Length,*StringOut);
    writeLine("stdout","The input string is: *StringIn");
    writeLine("stdout","The offset is *Offset and the length is *Length");
    writeLine("stdout","The output string is: *StringOut");
}
INPUT *StringIn="/tempZone/home/rods/sub1/foo1/", *Offset="10", *Length="4"
OUTPUT ruleExecOut
Core :: Workflow :: assign

```c
assign ( msParam_t * var,
         msParam_t * value )
```

**Parameters:**
[in] var - var is a msParam of type STR_MS_T which is a variable name or a Dollar Variable.
[in] value - value is a msParam of type STR_MS_T that is computed and value assigned to variable.

**Description:**
This micro-service assigns a value to a variable.

**Note:**
This micro-service is deprecated. In version 3.0, algebraic equations are used instead. Type checking is done to ensure consistency. Functions are provided to convert between data types, including:

- `str` convert integer to string variable
- `int` convert string to an integer
- `double` convert string to a double
- `bool` convert string to a Boolean variable

**Example Usage:**

```c
myTestRule {
    # Workflow command to assign a value to a variable
    # The assign microservice has been replaced with direct algebraic equations
    # Output from running the example is:
    # Value assigned is assign
    #
    # deprecated use:
    #   assign(*A,*B);
    #
    *A = *B;
    writeLine("stdout", "Value assigned is *A");
}
```

INPUT *B="assign"
OUTPUT ruleExecOut
break ( )

Parameters:
N/A.

Description:
This micro-service is used to break while, for and forEach loops.

Note:
This micro-service is similar to a break statement in the C language.

Example Usage:

myTestRule {
# Workflow command to break out of a loop
# Output from running the example is:
# abc
* A = list("a","b","c","d");
* B = "";
foreach(* A) {
if(* A == "d") then {
 break;
}
* B = * B ++ * A;
}
writeLine("stdout", * B);
}
INPUT null
OUTPUT ruleExecOut
4.69 Core :: Workflow :: cut

cut ( )

Parameters:
N/A.

Description:
This tells the rule engine to not retry any other applicable rules for this action.

Note:
The example invokes a "print" rule with two versions. The cut statement specifies that the second version will not be tried after the first version is explicitly failed.

Example:

myTestRule {
# Workflow operator to specify that no other versions of the rule will be tried
# Output from running the example is:
# ERROR: rcExecMyRule error. status = -1089000 CUT_ACTION_PROCESSED_ERR
# Level 0: DEBUG:
  print;
}

print {
  or {
    writeLine("serverLog", "print 1");
    cut;
    fail;
  }
  or {
    writeLine("serverLog", "print 2");
    succeed;
  }
}

INPUT null
OUTPUT ruleExecOut
4.70 Core :: Workflow :: delay

```
delay (        msParam_t * mPA ) {workflow :::: recovery}
```

**Parameters:**

<table>
<thead>
<tr>
<th>in</th>
<th>mPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>mPA</td>
<td>mPA is a msParam of type STR_MS_T which is a delayCondition about when to execute the body. These following tags are used:</td>
</tr>
<tr>
<td>EA</td>
<td>execAddress - host where the delayed execution needs to be performed</td>
</tr>
<tr>
<td>ET</td>
<td>execTime - absolute time when it needs to be performed</td>
</tr>
<tr>
<td>PLUS</td>
<td>relExeTime - relative to current time when it needs to execute</td>
</tr>
</tbody>
</table>
| EF | execFreq - frequency (in time widths) it needs to be performed. The format for EF is quite rich: 
  The EF value is of the format: nnnnU <directive> where nnnn is a number, and U is the unit of the number (s-sec,m-min,h-hour,d-day,y-year). The <directive> can be for the form: <empty-directive> - equal to REPEAT FOR EVER REPEAT UNTIL SUCCESS REPEAT nnnn TIMES - where nnnn is an integer REPEAT UNTIL <time> - where <time> is of the time format supported by checkDateFormat function below. REPEAT UNTIL SUCCESS OR UNTIL <time> REPEAT UNTIL SUCCESS OR nnnn TIMES DOUBLE FOR EVER DOUBLE UNTIL SUCCESS - delay is doubled every time. DOUBLE nnnn TIMES DOUBLE UNTIL <time> DOUBLE UNTIL SUCCESS OR UNTIL <time> DOUBLE UNTIL SUCCESS OR nnnn TIMES DOUBLE UNTIL SUCCESS UPTO <time> |

**Description:**

Execute a set of operations later when certain conditions are met. Can be used to perform periodic operations also. The set of operations are encapsulated in Brackets following the delay command. 

```
<PLUS>1m</PLUS><EF>10m</EF>
```

means start after 1 minute and repeat every 10 minutes

**Note:**

This micro-service is a set of statements that will be delayed in execution until delayCondition is true. The condition also supports repeating of the body until success or until some other condition is satisfied. This micro-service takes the delayCondition as the delay argument. The workflow is encapsulated in brackets, with the recovery micro-service inserted after the symbols " ::: " for each workflow micro-service. The delayCondition is given as a tagged condition. In the example, there are two conditions that are specified, one to specify execution after 30 seconds, and a second to repeat after 30 seconds. The iqstat command and iqdel commands can be used to delete the rule from the queue.

The command delayExec is deprecated. The micro-service "delayExec" is now equivalent to "delay". Both are interpreted using the new rule language syntax, with the workflow specified within brackets.
Example Usage:

myTestRule {
    # Workflow operator to execute a given workflow at a delayed specification
    # Input parameters are:
    # Delay condition composed from tags
    #    EA - host where the execution if performed
    #    ET - Absolute time when execution is done
    #    PLUSET - Relative time for execution
    #    EF - Execution frequency
    # Workflow specified within brackets
    # Output from running the example is:
    # exec
    # Output written to the iRODS/server/log/reLog log file:
    # writeLine: inString = Delayed exec
test = delay("<PLUSET>30s</PLUSET>") {
    writeLine("serverLog","Delayed exec");
}
writeLine("stdout","exec");

INPUT null
OUTPUT ruleExecOut
4.71 Core :: Workflow :: errorcode

errorcode ( micro-service)

Description:
The error return is trapped, allowing the rule to implement conditional processing of errors without having to invoke a recovery micro-service.

Parameters:
The argument is a micro-service that is being executed.

Description:
N/A.

Example Usage:

myTestRule {
# Workflow operator to trap an error code of passed command
# Input parameter is:
# micro-service whose error code will be trapped
# Output parameter is:
# none
if (errorcode( msiExecCmd(*Cmd, *Arg, "null", "null", "null", *Result)) < 0 ) {
   writeLine("stdout","Microservice execution had an error");
} else { writeLine("stdout","Microservice executed successfully"); }
}
INPUT *Cmd="hello", *ARG="iRODS"
OUTPUT ruleExecOut
4.72 Core :: Workflow :: fail

fail ( )

**Description:**
Fail immediately - recovery and retries are possible

**Parameters:**
N/A.

**Description:**
N/A.

**Example Usage:**

```c
myTestRule {
  # Workflow function to cause immediate failure
  # Output from running the example is:
  # ERROR: rcExecMyRule error. status = -1091000 FAIL_ACTION_ENCLOSED_ERR
  if(*A=="fail") {
    fail;
  }
}
INPUT *A="fail"
OUTPUT ruleExecOut
```
4.73  Core :: Workflow :: foreach

**foreach** (msParam_t * inlist) {workflow :: recovery}

**Parameters:**

[in]  inlist - a msParam of type STR_MS_T which is a comma separated string or StrArray_MS_T which is an array of strings or IntArray_MS_T which is an array of integers or GenQueryOut_MS_T which is an iCAT query result.

**Description:**
Performs a loop over a list of items given in different forms.

**Note:**
This executes a "for" loop in C-type language looping over a list. It takes a table (or list of strings, or comma-separated string list), and for each item in the list, executes the corresponding body of the for-loop. The first parameter specifies the variable that has the list (the same variable name is used in the body of the loop to denote an item of the list!). The workflow is a sequence of Micro-Services that is encapsulated in brackets, with the recovery procedure specified on each line after the " ::: " symbol.

The micro-service "forEachExec" is deprecated. It is replaced with "foreach".

**Example Usage:**

```c
myTestRule {
  # Workflow operator to iterate over a list
  # Input parameter is:
  # List
  # Workflow executed within brackets
  # Output from running the example is:
  # abcd
  *A = list("a","b","c","d");
  *B = "";
  foreach(*A) {
    *B = *B ++ *A;
  }
  writeLine("stdout", *B);
}
```

INPUT null
OUTPUT ruleExecOut
4.74 Core :: Workflow :: for

for (    msParam_t * initial,  
    msParam_t * condition,  
    msParam_t * step ) { workflow :: recovery }

Parameters:
[in] initial - a msParam of type STR_MS_T which is an initial assignment statement for the loop variable.
[in] condition - a msParam of type STR_MS_T which is a logical expression checking a condition.
[in] step - a msParam of type STR_MS_T which is an increment/decrement of loop variable.

Description:
It is a for loop in the rule language.

Note:
This micro-service loops over an integer *-variable until a condition is met. Similar to the "for" construct in C.

The micro-service "forExec" is deprecated and replaced with "for".

Example Usage:

myTestRule {
    # Input parameters are:
    # Loop initiation
    # Loop termination
    # Loop increment
    # Workflow in brackets
    # Output from running the example is:
    # abcd
    *A = list("a","b","c","d");
    *B = "";
    for(*I=0;*I<4;*I=*I+1) {
        *B = *B ++ elem(*A, *I);
    }
    writeLine("stdout", *B);
}

INPUT null
OUTPUT ruleExecOut
4.75  Core :: Workflow :: if

if (msParam_t * condition) {workflow ::: recovery }
else
 {workflow ::: recovery }

Parameters:
[in]  condition - a msParam of type STR_MS_T which is a logical expression computing to TRUE or FALSE.

Description:
This is an if-then-else construct in the rule language for conditional tests. If the logical expression is true, the specified workflow is executed. If the logical expression is false, the workflow after the "else" statement is executed.

Note:
The argument is a conditional check. If the check is successful (TRUE), the micro-service sequence in the workflow will be executed. If the check fails, then the micro-service sequence after the "else" statement will be executed.

The micro-service "ifExec" is deprecated and replaced with "if".

Example Usage:

myTestRule {
 # Workflow operator to evaluate conditional expression
 # Input parameters are:
 # Logical expression that computes to TRUE or FALSE
 # Workflow to be executed defined within brackets
 # Else clause defined within brackets
 # Output from running the example is:
 # 0
 if(*A=="0") {
   writeLine("stdout", "0");
 } else {
   writeLine("stdout", "not 0");
 }
}
INPUT *A="0"
OUTPUT ruleExecOut
4.76 Core :: Workflow :: applyAllRules

```c
applyAllRules ( msParam_t * actionParam,
                msParam_t * reiSaveFlagParam,
                msParam_t * allRuleExecFlagParam )
```

**Parameters:**
- **[in]** actionParam - a msParam of type STR_MS_T which is the name of an action to be executed.
- **[in]** reiSaveFlagParam - a msParam of type STR_MS_T which is 0 or 1. The value is used to check if the rei structure needs to be saved at every rule invocation inside the execution. This helps to save time if the rei structure is known not to be changed when executing the underlying rules.
- **[in]** allRuleExecFlagParam - allRuleExecFlagParam is a msParam of type STR_MS_T which is 0 or 1 specifies whether the "apply all rule" condition applies only to the actionParam invocation or is recursively done at all levels of invocation of every rule inside the execution.

**Description:**
This micro-service executes all applicable rules for a given action name.

**Note:**
Normal operations of the rule engine is to stop after a rule (one of the alternate actions) completes successfully. But in some cases, one may want the rule engine to try all alternatives and succeed in as many as possible. Then by firing that rule under this micro-service all alternatives are tried.

The actionParam name should not be quoted in the micro-service invocation.

**Example Usage:**

```c
myTestRule {
    # Input parameters are:
    # Action to perform
    # Flag for whether to save REI structure, 1 is yes
    # Flag for whether to apply recursively, 1 is yes
    # Output from executing the example is:
    # print 1
    # print 2
    applyAllRules(print, *SaveREI, *All);
}
print{
    or {
        writeLine("stdout", "print 1");
    }
    or {
        writeLine("stdout", "print 2");
    }
}
INPUT *All="1", *SaveREI="0"
OUTPUT ruleExecOut
```
4.77  Core :: Workflow :: msiGoodFailure

msiGoodFailure (

Parameters:
N/A.

Description:
This micro-service performs no operations but fails the current rule application immediately even if the body still has some more micro-services to execute. Other definitions of the rule are not retried upon this failure. It is useful when you want to fail and ensure no recovery is initiated.

Note:
Useful when you want to fail a rule without retries.

Example Usage:

myTestRule {
  # Workflow function to fail immediately with no recovery
  # Output from running the example is:
  # ERROR: rcExecMyRule error. status = -1088000 RETRY_WITHOUT_RECOVERY_ERR
  msiGoodFailure;
}
INPUT null
OUTPUT null
4.78 Core :: Workflow :: msiSleep

msiSleep ( msParam_t * secPtr,  
            msParam_t * microsecPtr )

Parameters:
[in] secPtr - secPtr is a msParam of type STR_MS_T which is seconds
[in] microsecPtr - microsecPtr is a msParam of type STR_MS_T which is microseconds

Description:
Sleep for some amount of time

Note:
Similar to sleep in C

Example Usage:

myTestRule {
    # Input parameters are:
    # Number of seconds to sleep
    # Number of micro-seconds to sleep
    # Output from running the example is:
    # Jun 01 2011 17:04:59
    # Jun 01 2011 17:05:09
        writeLine("stdout", timestr(time()));
        msiSleep(*Sec, *MicroSec);
        writeLine("stdout", timestr(time()));
    }
INPUT *Sec="10", *MicroSec="0"
OUTPUT ruleExecOut
Core :: Workflow :: nop, null

nop, null - No action

Parameters:
N/A.

Description:
Executes "no action" or "no operation".

Example usage:

myTestRule {
    # Workflow function for no operation
    # Output from running the example is:
    # nop
    nop;
    writeln("stdout", "nop");
}

INPUT null
OUTPUT ruleExecOut
Print_hello ( )

Parameters: None

Description: Prints out the string "Hello" to stdout.

Note: This executes the "hello" command stored in the server/bin/cmd directory. A recovery micro-service is available called "recover_print_hello".

Example usage:

myTestRule {
    # Output string is written to stdout
    writeLine("stdout","Execute command to print out hello");
    print_hello;
}

INPUT null
OUTPUT ruleExecOut
4.81 Core :: Workflow :: remote

Remote (msParam_t * mPD, msParam_t * mPA) {workflow :: recovery}

Parameters:
[in] mPD - a msParam of type STR_MS_T which is a host name of the server where
the body needs to be executed.
[in] mPA - a msParam of type STR_MS_T which is a delayCondition about when to
execute the body.

Description:
Manages the execution of a set of micro-services at a remote location.

Note:
This micro-service takes a set of micro-services that need to be executed at a remote iRODS server. The
execution is done immediately and synchronously with the result returned back from the call.

The micro-service "remoteExec" is deprecated and replaced with "remote".

Example Usage:

myTestRule {
  # Workflow operation to execute microservices at a remote location
  # Input parameters are:
  #  Host name where workflow is executed
  #  Delaycondition for executing the workflow
  #  Workflow :::: recovery-workflow that will be executed, listed in brackets
  # Output from running the example written to server log:
  #  writeLine: inString = local exec
  #  writeLine: inString = remote exec
  # Output from running the example written to standard out:
  #  local exec
  writeLine("serverLog","local exec");
  remote("localhost", "null") {
    writeLine("serverLog","remote exec");
  }
  writeLine("stdout", "local exec");
}
INPUT null
OUTPUT ruleExecOut
4.82 Core :: Workflow :: succeed

**succeed** - Succeed immediately

**Parameters:**
N/A.

**Description:**
Succeed immediately.

**Example usage:**

```java
myTestRule {
    # Workflow operation to cause rule to immediately succeed
    # Output from running the example is:
    # succeed
    if(*A == "succeed") {
        writeLine("stdout", "succeed");
        succeed;
    } else {
        fail;
    }
}
INPUT *A="succeed"
OUTPUT ruleExecOut
```
4.83 Core :: Workflow :: while

while ( msParam_t * condition ) {workflow :::: recovery }

Parameters:
[in] condition - a msParam of type STR_MS_T which is a logical expression computing to TRUE or FALSE.

Description:
This is a while loop in the rule language.

Note:
The first argument is a condition that will be checked on each loop iteration. The body of the while loop, given as a sequence of micro-services :::: recovery-micro-service, is listed in brackets.

The micro-service "whileExec" is deprecated and replaced with "while".

Example Usage:

myTestRule {
  # Workflow operation to loop until condition is false
  # Input parameter is
  # Logical expression which evaluates to TRUE or FALSE
  # Workflow that is executed, defined within brackets
  # Output from running the example is:
  # abcd
  *A = list("a","b","c","d");
  *B = "";
  *I=0;
  while(*I < 4) {
    *B = *B ++ elem(*A, *I);
    *I = *I + 1;
  }
  writeLine("stdout", *B);
}

INPUT null
OUTPUT ruleExecOut
### 4.84 Core :: Workflow :: writeLine

**writeLine**

```c
(msParam_t * where,
 msParam_t * inString)
```

**Parameters:**
- **where**
  - A `msParam` of type `STR_MS_T` which is the buffer name in `ruleExecOut`. Currently `stdout`, `stderr`, and `serverLog` can be used.
- **inString**
  - A `msParam` of type `STR_MS_T` which is a string to be written into a buffer.

**Description:**
This micro-service writes a given string followed by a new-line character into the target buffer in `ruleExecOut` Parameter.

**Note:**
This micro-service takes a given buffer string and appends it to the back of the buffer (either `stdout` or `stderr` or `serverLog` in `ruleExecOut` parameter) followed by a new line character. In the OUTPUT line, the `ruleExecOut` is a system MS-parameter (*variable) that is automatically available.

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  # Name of output buffer
  # stdout
  # stderr
  # serverLog
  # String to write
  # Output from running the example is:
  # line
  writeLine(*Where, *StringIn);
}
```

INPUT *Where="stdout", *StringIn="line"
OUTPUT ruleExecOut
4.85  Core :: Workflow :: writeString

writeString  ( msParam_t *  where,
                 msParam_t *  inString )

Parameters:
[in]  where        - where is a msParam of type STR_MS_T which is the buffer name
                 in ruleExecOut. Currently stdout and stderr.
[in]  inString     - inString is a msParam of type STR_MS_T which is a string to be
                 written into the buffer

Description:
This micro-service writes a given string into the target buffer in ruleExecOut parameter.

Note:
This micro-service takes a given buffer string and appends it to the back of the buffer (either stdout or
stderr or serverLog). In the OUTPUT line, the ruleExecOut is a system MS-parameter (*variable) that is
automatically available that specifies copying of the "stdout" buffer to the client.

Example Usage:

myTestRule {  
    # Input parameters are:
    # Buffer where the string is written
    #  stdout
    #  stderr
    #  serverLog
    # String that is written
    # Output from running the example is:
    # string
        writeString(*Where, *StringIn);
        writeLine(*Where,"cheese");
    }
INPUT *Where="stdout", *StringIn="string"
OUTPUT ruleExecOut
4.86 Core :: XMessaging System :: msiCreateXmsgInp

**msiCreateXmsgInp** (msParam_t * inMsgNumber,
                        msParam_t * inMsgType,
                        msParam_t * inNumberOfReceivers,
                        msParam_t * inMsg,
                        msParam_t * inNumberOfDeliverySites,
                        msParam_t * inDeliveryAddressList,
                        msParam_t * inDeliveryPortList,
                        msParam_t * inMiscInfo,
                        msParam_t * inXmsgTicketInfoParam,
                        msParam_t * outSendXmsgInpParam)

**Parameters:**

- **in**
  - `inMsgNumber`: a msParam of type uint or STR_MS_T which is a message serial number.
  - `inMsgType`: a msParam of type uint or STR_MS_T which is currently 0 (SINGLE_MSG_TICKET) or 1 (MULTI_MSG_TICKET).
  - `inNumberOfReceivers`: a msParam of type uint or STR_MS_T which is a number of receivers of the message.
  - `inMsg`: a msParam of type STR_MS_T which is a message body.
  - `inNumberOfDeliverySites`: a msParam of type int or STR_MS_T which is a Number of Receiving Addresses.
  - `inDeliveryAddressList`: a msParam of type STR_MS_T which is a List of Host Addresses (comma separated).
  - `inDeliveryPortList`: a msParam of type STR_MS_T which is a List of Corresponding Ports (comma separated).
  - `inMiscInfo`: a msParam of type STR_MS_T which is other Information.
  - `inXmsgTicketInfoParam`: a msParam of type XmsgTicketInfo_MS_T which is `outXmsgTicketInfoParam` from msiXmsgCreateStream

- **out**
  - `outSendXmsgInpParam`: a msParam of type SendXmsgInp_MS_T which is a Xmsg packet.

**Description:**

Given all input information values this micro-service creates an Xmsg packet.

**Note:**

The Xmsg packet can be sent using the msiSendXmsg micro-service.

**Example Usage:**

```c
myTestRule {
    # Input parameters are:
    #  Message serial number
    #  Message type
    #    0 SINGLE_MSG_TICKET
    #    1 MULTI_MSG_TICKET
    #  Number of receivers for message
    #  Message body
    #  Number of receiving sites, comma separated
    #  List of Host Addresses
    #  List of Ports
    #  Miscellaneous information
    #  outXmsgTicketInfoParam from msiXmsgCreateStream
    # Output parameter is:
    #  Xmsg packet
    for (*I = 0 ; *I < *Count ; *I = *I + 1)
```
{ 
  msiXmsgServerConnect(*Conn);
  msiXmsgCreateStream(*Conn,*A,*Tic);
  msiCreateXmsgInp("1","1","1","TTTest\*1","0","",","",*Tic,*MParam);
  msiSendXmsg(*Conn,*MParam);
  msiXmsgServerDisConnect(*Conn);

  # now read the message that was sent. The read can be done from a remote server.
  msiXmsgServerConnect(*Conn1);
  msiRcvXmsg(*Conn1,*Tic,"1",*MType,*MMsg,*MSender);
  writeLine("stdout",*MMsg);
  msiXmsgServerDisConnect(*Conn1);
}
}

INPUT *A=100, *Count=10
OUTPUT ruleExecOut,*MSender,*Mtype,*MMsg
4.87  Core :: XMessaging System  :: msiRcvXmsg

msiRcvXmsg ( msParam_t * inConnParam,
             msParam_t * inTicketNumber,
             msParam_t * inMsgNumber,
             msParam_t * outMsgType,
             msParam_t * outMsg,
             msParam_t * outSendUser )

Parameters:

[in]  inConnParam  - a msParam of type RcComm_MS_T which is a connection descriptor obtained by msiXmsgServerConnect.

[in]  inTicketNumber  - a msParam of type XmsgTicketInfo_MS_T or STR_MS_T or unit which is an outXmsgTicketInfoParam from msiXmsgCreateStream or outXmsgTicketInfoParam->rcvTicket (a string which the sender passes to the receiver).

[in]  inMsgNumber  - a msParam of type unit or STR_MS_T which is a message serial number to fetch.

[out]  outMsgType  - a msParam of type STR_MS_T which is a message type.

[out]  outMsg  - a msParam of type STR_MS_T which is a message body.

[out]  outSendUser  - a msParam of type STR_MS_T which is the sender information.

Description:

This micro-service receives an Xmsg packet.

Note:

Receives an X message packet using the connection made by msiXmsgServerConnect.

Example Usage:

myTestRule {  
  # Input parameters are:
  # Connection descriptor from msiXmsgServerConnect
  # OutXmsgTicketInfoParam from msiXmsgCreateStream
  # Message serial number
  # Output parameters are:
  # Message type
  # Message Body
  # Sender
  # Output from running the example is:
  # TTTest0
  # TTTest1
  # TTTest2
  # TTTest3
  for (*I = 0 ; *I < *Count ; *I = *I + 1) {  
    # send a message
    msiXmsgServerConnect(*Conn);
    msiXmsgCreateStream(*Conn,*A,*Tic);
    msiCreateXmsgInp("1","0","1","TTTest*I","0","" ,"" ,*Tic,*MParam);
    msiSendXmsg(*Conn,*MParam);
    msiXmsgServerDisConnect(*Conn);
    # now receive the message
    msiXmsgServerConnect(*Conn1);
    msiRcvXmsg(*Conn1,*Tic,"1",*MType,*MMsg,*MSender);
    writeLine("stdout","MMsg");  
  }
}
msiXmsgServerDisConnect(*Conn1);
}
}
INPUT *A=100, *Count= 4
OUTPUT ruleExecOut
4.88 Core :: XMessaging System :: msiSendXmsg

msiSendXmsg ( msParam_t * inConnParam,
               msParam_t * inSendXmsgInpParam )

Parameters:
[in] inConnParam - a msParam of type RcComm_MS_T which is a connection descriptor obtained by msiXmsgServerConnect.
[in] inSendXmsgInpParam - a msParam of type SendXmsgInp_MS_T outSendXmsgInpParam from inSendXmsgInpParam.

Description:
This micro-service sends an Xmsg packet.

Note:
Sends an Xmsg packet created by msiCreateXmsgInp using the connection made by msiXmsgServerConnect

Example Usage:

myTestRule {
    # Input parameters are:
    # Connection descriptor from msiXmsgServerConnect
    # OutXmsgTicketInfoParam from msiXmsgCreateStream
    # Output from running the example is:
    # TTTTest0
    # TTTTest1
    # TTTTest2
    # TTTTest3
    for (*I = 0 ; *I < *Count ; *I = *I + 1) {
        # send a message
        msiXmsgServerConnect(*Conn);
        msiXmsgCreateStream(*Conn,*A,*Tic);
        msiCreateXmsgInp("1","0","1","TTTest*I","0","","",*Tic,*MParam);
        msiSendXmsg(*Conn,*MParam);
        msiXmsgServerDisConnect(*Conn);

        # now receive the message
        msiXmsgServerConnect(*Conn1);
        msiRxMsg(*Conn1,*Tic,"1","*MType,*MMsg,*MSender);
        writeLine("stdout",*MMsg);
        msiXmsgServerDisConnect(*Conn1);
    }
}  

INPUT *A=100, *Count= 4
OUTPUT ruleExecOut
4.89 Core :: XMessaging System :: msiXmsgCreateStream

msiXmsgCreateStream (msParam_t * inConnParam,
                        msParam_t * inGgetXmsgTicketInpParam,
                        msParam_t * outXmsgTicketInfoParam )

Parameters:
[in] inConnParam - a msParam of type RcComm_MS_T which is a connection descriptor obtained by msiXmsgServerConnect.
[in] inGgetXmsgTicketInpParam - a msParam of type GetXmsgTicketInp_MS_T which is actually an integer giving expiration time.
[out] outXmsgTicketInfoParam - a msParam of type XmsgTicketInfo_MS_T which is an information struct for the ticket generated for this stream.

Description:
This micro-service creates a new Message Stream.

Note:
A new message stream is created such that the process (or any other process) can send messages.

Example Usage:

myTestRule {
    # Input parameters are:
    # Connection descriptor from msiXmsgServerConnect
    # Message expiration time in seconds
    # Output parameters are:
    # OutXmsgTicketInfoParam information structure
    # Output from running the example is:
    # TTTes0
    # TTTes1
    # TTTes2
    # TTTes3
    for (*I = 0 ; *I < *Count ; *I = *I + 1) {
        # send a message
        msiXmsgServerConnect(*Conn);
        msiXmsgCreateStream(*Conn,*A,*Tic);
        msiCreateXmsgIHp("1","0","1","TTTesI","0","" ,"" ,"" ,*Tic,*MParam);
        msiSendXmsg(*Conn,*MParam);
        msiXmsgServerDisConnect(*Conn);

        # now receive the message
        msiXmsgServerConnect(*Conn1);
        msiRecXmsg(*Conn1,*Tic,"1",*MType,*MMsg,*MSender);
        writeLine("stdout",*MMsg);
        msiXmsgServerDisConnect(*Conn1);
    }
}
INPUT *A=100, *Count= 4
OUTPUT ruleExecOut
4.90  Core :: XMessaging System :: msiXmsgServerConnect

msiXmsgServerConnect ( msParam_t * outConnParam )

Parameters:
[out] outConnParam - a msParam of type RcComm_MS_T which is a connection descriptor.

Description:
Connect to the XMessage server.

Note:
This micro-service connects to the XMessage Server as designated in the iRODS Environment file.

Example Usage:

myTestRule {
# Output parameter is:
# Connection descriptor
# Output from running the example is:
# TTTest0
# TTTest1
# TTTest2
# TTTest3
for (*I = 0 ; *I < *Count ; *I = *I + 1) {
    # send a message
    msiXmsgServerConnect(*Conn);
    msiXmsgCreateStream(*Conn,*A,*Tic);
    msiCreateXmsgInp("1","0","1","TTTest*I","0" ,"" ,"" ,"" ,*Tic,*MParam);
    msiSendXmsg(*Conn,*MParam);
    msiXmsgServerDisConnect(*Conn);

    # now receive the message
    msiXmsgServerConnect(*Conn1);
    msiRcvXmsg(*Conn1,*Tic,1" ,*MType,*MMsg,*MSender);
    writeLine("stdout",*MMsg);
    msiXmsgServerDisConnect(*Conn1);
}
}

INPUT *A=100, *Count= 4
OUTPUT ruleExecOut
4.91 Core :: XMessaging System :: msiXmsgServerDisConnect

msiXmsgServerDisConnect ( msParam_t * inConnParam )

Parameters:
[in] inConnParam - a msParam of type RcComm_MS_T which is a connection descriptor obtained by msiXmsgServerConnect.

Description:
This micro-service disconnects from the XMessage Server.

Note:
Disconnects a connection made by msiXmsgServerConnect

Example Usage:

myTestRule {
  # Input parameter is:
  # Connection descriptor from msiXmsgServerConnect
  # Output from running the example is:
  #  TTTTest0
  #  TTTTest1
  #  TTTTest2
  #  TTTTest3
  for (*I = 0 ; *I < *Count ; *I = *I + 1) {
    # send a message
    msiXmsgServerConnect(*Conn);
    msiXmsgCreateStream(*Conn,*A,*Tic);
    msiCreateXmsgInp("1","0","1","TTTest*","0","","",*Tic,*MParam);
    msiSendXmsg(*Conn,*MParam);
    msiXmsgServerDisConnect(*Conn);

    # now receive the message
    msiXmsgServerConnect(*Conn1);
    msiRcvXmsg(*Conn1,*Tic,"1",*MType,*MMsg,*MSender);
    writeLine("stdout",*MMsg);
    msiXmsgServerDisConnect(*Conn1);
  }
}

INPUT *A=100, *Count= 4
OUTPUT ruleExecOut
4.92 Core :: XMessaging System :: readXMsg

readXMsg (msParam_t * inStreamId,
         msParam_t * inCondRead,
         msParam_t * outMsgNum,
         msParam_t * outSeqNum,
         msParam_t * outHdr,
         msParam_t * outMsg,
         msParam_t * outUser,
         msParam_t * outAddr )

Parameters:
[in] inStreamId - of type STR_MS_T or INT_MAS_T - the XMsg streamId
number possibly generated by a msiXmsgCreateStream micro-
service or a supported standard stream with ids 1 thru 5
[in] inCondRead - of type STR_MS_T - boolean condition for a packet to satisfy
and the first packet that satisfies the condition is read from the
XMsg Stream
[out] outMsgNum - of type INT_MS_T - message number of the incoming packet (as
given by message source)
[out] outSeqNum - of type INT_MS_T - sequence number of the incoming packet
(as given by Xmsg Server)
[out] outHdr - of type STR_MS_T - header string of the incoming message packet
[out] outMsg - of type STR_MS_T - message string of the incoming message packet
[out] outUser - of type STR_MS_T - userName of the sender of the packet
[out] outAddr - of type STR_MS_T - address of the sending site of the packet
(host address and process-id)

Description:
This micro-service reads a message packet from an XMsgStream

Note:
This micro-service reads into buffer a message packet from the XMsg Server

Example Usage:

myTestRule {
  # Input parameters are:
  # Stream ID
  # Condition for reading a packet
  #  *XADDR != "srbbrick14:20135" - form host:process-id
  #  *XUSER != "rods@tempZone" - form user@zone
  #  *XHDR == "header" - header of the message
  #  *XMISC == "msic_infor" - miscellaneous message part
  #  *XTIME - Unix time in seconds from Jan 1, 1970
  # Output parameters are:
  # Message number
  # Sequence number from the Xmsg Server
  # Header string
  # Message
  # Sender
  # Sending site
  # Output from running the example is:
  # Sequence Number is 0
  # Message Header is Test
  # Message is Body of the message

}
# Sender is rods@tempZone
# Address is reagan-VirtualBox:4078 (this may change with each run)
writeXMsg(*StreamID,*Header,*Message);
# now read the message
readXMsg(*StreamID,*Condition,*MessageNum,*SequenceNum,*MsgHeader,*MsgMessage,*Sender,*Address);
writeLine("stdout","Sequence Number is *SequenceNum");
writeLine("stdout","Message Header is *MsgHeader");
writeLine("stdout","Message is *MsgMessage");
writeLine("stdout","Sender is *Sender");
writeLine("stdout","Address is *Address");
}
INPUT *StreamID="1",*Header="Test",*Message="Body of the message",*Condition='"*XUSER == "rods@tempZone"",&*MessageNum="1"
OUTPUT ruleExecOut
4.93  Core :: XMessaging System  :: writeXMsg

writeXMsg ( msParam_t * inStreamId,  
            msParam_t * inHdr,  
            msParam_t * inMsg )

Parameters:
[in] inStreamId - of type STR_MS_T or INT_MAS_T - the XMsg streamId  
                   number possibly generated by a msiXmsgCreateStream micro-
                   service or a supported standard stream with ids 1 through 5  
[in] inHdr      - of type STR_MS_T - header string to be sent in the message packet  
[in] inMsg      - of type STR_MS_T - message string to be sent in the message packet

Description:
This micro-service writes a given string into an XMsgStream

Note:
This micro-service takes a given buffer string and sends it as a message packet to the XMsg Server

Example Usage:

myTestRule {  
    # Input parameters are:  
    #  Stream ID  
    #  Header  
    #  Message  
    # Output from running the example is:  
    #  Sequence Number is 0  
    #  Message Header is Test  
    #  Message is Body of the message  
    #  Sender is rods@tempZone  
    #  Address is reagan-VirtualBox:4078 (this may change with each run)  

    writeXMsg(*StreamID,*Header,*Message);  

    # now read the message  
    readXMsg(*StreamID,*Condition,*MessageNum,*SequenceNum,*MsgHeader,*MsgMessage,*Sender,*Address);  
    writeLine("stdout","Sequence Number is *SequenceNum");  
    writeLine("stdout","Message Header is *MsgHeader");  
    writeLine("stdout","Message is *MsgMessage");  
    writeLine("stdout","Sender is *Sender");  
    writeLine("stdout","Address is *Address");  
}

INPUT *StreamID="1",*Header="Test",*Message="Body of the message",*Condition="XUSER == "rods@tempZone"",&*MessageNum="1"  
OUTPUT ruleExecOut
4.94  Core :: Framework Services System :: msiCheckHostAccessControl

msiCheckHostAccessControl (   )

Parameters:
None

Description:
This micro-service sets the access control policy. It checks the access control by user and group from a given host based on the policy given in the HostAccessControl file.

Note:
The policy is implemented in the core.re file.

This micro-service controls access to the iRODS service based on the information in the host based access configuration file: iRODS/server/config/HostAccessControl. This is a column-based file that identifies who is allowed to connect if the acChkHostAccessControl policy is turned on.

    The first column specifies a user that is allowed to connect to this iRODS server. An entry of "all" means all users are allowed.

    The second column specifies the group name. An entry of "all" means, all groups are allowed.

    The third and fourth columns specify the address and the address mask. Together, they define the client IP addresses/domains that are permitted to connect to the iRODS server. The address column specifies the IP address and the Mask column specifies which bits will be ignored, i.e., after those bits are taken out, the connection address must match the address in the address column.

        <name>   <group>   <address>     <mask>
        all             all            127.0.0.1       255.255.255.255

Example Usage:

acChkHostAccessControl {
    # No arguments
    # The file iRODS/server/config/HOST_ACCESS_CONTROL_FILE
    # is read to identify hosts that can access iRODS.
    msiCheckHostAccessControl;
}
4.95 Core :: Framework Services System :: msiDeleteDisallowed

msiDeleteDisallowed ( )

Parameters:
None

Description:
This micro-service sets the policy for specifying that certain data cannot be deleted.

Note:
The policy is implemented in the core.re file. An acDataDeletePolicy rule condition is used to decide which collections to protect. The output that is generated when you try to delete a protected file is:
ERROR: rmUtil: rm error for /tempZone/home/rods/sub1/foo3, status = -1097000 status = -1097000 NO_RULE_OR_MSI_FUNCTION_FOUND_ERR

Example Usage:

acDataDeletePolicy {
  # Output when try to delete a file:
  # ERROR: rmUtil: rm error for /tempZone/home/rods/sub1/foo3, status = -1097000 status = -1097000
  NO_RULE_OR_MSI_FUNCTION_FOUND_ERR
  # Rule condition is used to choose which collections to protect
  ON($objPath like "/tempZone/home/rods/*/"") {
    msiDeleteDisallowed;
  }
}


4.96 Core :: Framework Services System :: msiDigestMonStat

msiDigestMonStat ( msParam_t * cpu_wght,
                msParam_t * mem_wght,
                msParam_t * swap_wght,
                msParam_t * runq_wght,
                msParam_t * disk_wght,
                msParam_t * netin_wght,
                msParam_t * netout_wght )

Parameters:
[in] cpu_wght - Required - a msParam of type STR_MS_T defining relative CPU weighting.
[in] mem_wght - Required - a msParam of type STR_MS_T defining relative memory weighting
[in] swap_wght - Required - a msParam of type STR_MS_T defining relative swap weighting
[in] runq_wght - Required - a msParam of type STR_MS_T defining relative run queue weighting
[in] disk_wght - Required - a msParam of type STR_MS_T defining relative disk space weighting
[in] netin_wght - Required - a msParam of type STR_MS_T defining relative inbound network weighting
[in] netout_wght - Required - a msParam of type STR_MS_T defining relative outbound network weighting

Description:
This micro-service calculates and stores a load factor for each connected resource based on the weighting values passed in as parameters.

Note:
The following values are loaded from R_LOAD_SERVER:
  cpu_used
  mem_used
  swap_used
  runq_load
  disk_space
  net_input
  net_output
The stored load factor is calculated as such:
load_factor = cpu_wght*cpu_used + mem_wght*mem_used + swap_wght*swap_used +
runq_wght*runq_load + disk_wght*disk_space + netin_wght*net_input + netout_wght*net_output

The digest of the load factor can be retrieved by the iquest query:
  iquest "SELECT SLD_RESC_NAME,SLD_LOAD_FACTOR"
See also: https://www.irods.org/index.php/Resource_Monitoring_System

Example Usage:

myTestRule {
  # Input parameters are:
  # CPU weight
  # Memory weight
  # Swap weight
  # Run queue weight
  # Disk weight
# Network transfer in weight
# Network transfer out weight
# Output from running the example is:
# CPU weight is 1, Memory weight is 1, Swap weight is 0, Run queue weight is 0
# Disk weight is 0, Network transfer in rate is 1, Network transfer out rate is 1
# List of resources and the computed load factor digest
writeLine("stdout","CPU weight is *Cpuw, Memory weight is *Memw, Swap weight is *Swapw, Run
queue weight is *Runw");
writeLine("stdout","Disk weight is *Diskw, Network transfer in rate is *Netinw, Network transfer out rate
is *Netow");
msiExecStrCondQuery("SELECT SLD_RESC_NAME,SLD_LOAD_FACTOR",*QOut);
foreach(*QOut) { msiPrintKeyValPair("stdout",*QOut) }
}

INPUT *Cpuw="1", *Memw="1", *Swapw="0", *Runw="0", *Diskw="0", *Netinw="1", *Netow="1"
OUTPUT ruleExecOut
4.97  Core :: Framework Services System :: msiFlushMonStat

 MSI_FLUSH_MON_STAT (
    msParam_t * inpParam1,
    msParam_t * inpParam2 )

Parameters:

[in]  inpParam1 - Required - a msParam of type STR_MS_T defining the timespan in hours. "default" is equal to 24 hours.
[in]  inpParam2 - Required - a msParam of type STR_MS_T defining the tablename to be flushed. Currently must be either "serverload" or "serverloaddigest".

Description:
This micro-service flushes the servers' monitoring statistics.

Note:
This micro-service removes the servers' metrics older than the number of hours in "timespan".
See also: https://www.irods.org/index.php/Resource_Monitoring_System

Example Usage:

myTestRule {
    # Input parameters are:
    # Timespan before which stats are deleted (in hours)
    # Table to be flushed
    # serverload
    # serverloaddigest
    # Output from running the example is a list of load factors per resource
    msiFlushMonStat(*Time, *Table);
    msiExecStrCondQuery("SELECT SLD_RESC_NAME, SLD_LOAD_FACTOR", *QOut);
    foreach(*QOut) { msiPrintKeyValPair("stdout", *QOut); }
}

INPUT *Time="24", *Table="serverload", *Cpuw="1", *Memw="1", *Swapw="0", *Runw="0",
*Diskw="0", *Netinw="1", *Netow="1"
OUTPUT ruleExecOut
4.98 Core :: Framework Services System :: msiListEnabledMS

msiListEnabledMS (msParam_t * outKVPairs)

Parameters:
[out] outKVPairs - A KeyValPair_MS_T containing the results.

Description:
Returns the list of compiled micro-services on the local iRODS server

Note:
This micro-service looks at reAction.h and returns the list of compiled micro-services on the local iRODS server. The results are written to a KeyValPair_MS_T. For each pair the keyword is the MS name while the value is the module where the micro-service belongs. Standard non-module micro-services are listed as "core".

Example Usage:

myTestRule {
  # Output
  # Buffer holding list of microservices in form Key=Value
  # Output from running the example is:
  # List of microservices that are enabled
  msiListEnabledMS(*Buf);
  writeKeyValPairs("stdout",*Buf,":");
}

INPUT null
OUTPUT ruleExecOut
4.99 Core :: Framework Services System :: msiNoChkFilePathPerm

msiNoChkFilePathPerm ( )

Parameters:
None

Description:
This micro-service turns off the check on file path permissions when registering a file.

Note:
This micro-service sets the policy for checking the file path permission when registering a physical file path using commands such as ireg. This rule also sets the policy for checking the file path when unregistering a data object without deleting the physical file. Normally, a normal user cannot unregister a data object if the physical file is located in a resource vault. The msiNoChkFilePathPerm allows this check to be bypassed.

WARNING - This function can create a security problem if used incorrectly.

Example Usage:

acNoChkFilePathPerm {
    # No arguments
    msiNoChkFilePathPerm;
}
4.100 Core :: Framework Services System :: msiServerBackup

msiServerBackup ( msParam_t *options,
                  msParam_t *keyValOut )

Parameters:
[in] options - Optional - a STR_MS_T that contains one or more options in the format
keyWd1=value1++++keyWd2=value2++++keyWd3=value3...
This is a placeholder for now.
[out] keyValOut - a KeyValPair_MS_T with the number of files and bytes written.

Description:
This micro-service copies the iRODS server files to the local vault and registers them into iRODS. Object files (.o) and binaries are not included.

Note:
The server backup does not include the /Vault files or the database.
The .irodsEnv file and the .irodsA files are not included. They can be recreated by running ./irodssetup.
The iRODS server files are written into the iRODS directory:
    /zone/home/rods-admin/system_backups/system-name_date/iRODS
where your zone name, your rods-admin name, and your system-name are used in the path name.
The iRODS server files can then be bundled into a tar file, and replicated to another data grid through
iRODS commands.

Example Usage:

myTestRule {
    # Input parameter is:
    # Options - currently none are specified for controlling server backup
    # Output parameter is:
    # Result - a keyvalpair structure holding number of files and size
    #
    # This will take a while to run.
    # Backup files are stored in a directory as hostname_timestamp:
    #
    # $ ils system_backups
    # /tempZone/home/rods/system_backups:
    # C- /tempZone/home/rods/system_backups/localhost_2011-08-19.16:00:29
    #
    msiServerBackup(*Opt,*Result);
    writeKeyValPairs("stdout",*Result, " : ");
}
INPUT *Opt=""
OUTPUT ruleExecOut
### 4.101 Core :: Framework Services System :: msiSysMetaModify

```c
msiSysMetaModify ( msParam_t * sysMetadata,
                    msParam_t * value )
```

**Parameters:**

- **sysMetadata** - A STR_MS_T which specifies the system metadata to be modified. Allowed values are: "datatype", "comment", "time".
- **value** - A STR_MS_T which specifies the value to be given to the system metadata.

**Description:**

Modify system metadata.

**Note:**

This call should only be used within a core.re policy, as it requires that the rei structure be initialized for file manipulation.

**Example Usage:**

```c
acPostProcForPut { 
    ON($filePath like "*.txt") { 
        msiSysMetaModify("datatype","text"); 
    } 
} 
```
4.102 Core :: Framework Services System :: msiNoTrashCan

msiNoTrashCan()

Parameters:
None

Description:
This micro-service sets the policy to no trash can.

Note:
The default policy is that a trash can will be used. When a file is deleted from iRODS, it is actually moved to the trash can located in a corresponding path under /data-grid/trash. With no trash can, instead the file is deleted directly. Moving the file to the trash can is normally much faster, but then the trash can should be periodically emptied.

Example Usage:

acTrashPolicy {
  # System control
  msiNoTrashCan;
}

4.103 Core :: Framework Services System :: msiOprDisallowed

msiOprDisallowed ( )

**Parameters:**
None

**Description:**
This generic micro-service sets the policy for determining that the desired action is not allowed.

**Note:**
The msiOprDisallowed microservice can be used by all the rules to disallow the execution of specific actions.

**Example Usage:**

```plaintext
acSetRescSchemeForCreate {
  ON (ObjPath like "\*foo*") {
    msiOprDisallowed;
  }
}
```
4.104 Core :: Framework Services System :: msiServerMonPerf

msiServerMonPerf ( msParam_t* verb,  
                    msParam_t* ptime )

Parameters:
[in] verb - a msParam of type STR_MS_T defining verbose mode:
    "default" - not verbose
    "verbose" - verbose mode
[in] ptime - a msParam of type STR_MS_T defining probe time in seconds. "default"
             is equal to 10 seconds.

Description:
This micro-service monitors the servers' activity and performance.

Note:
This micro-service monitors the servers' activity and performance for CPU, network, memory and more. It
retrieves the list of servers to monitor from the MON_CFG_FILE if it exists, or the iCAT if the
configuration file does not exist.
The MON_PERF_SCRIPT is executed on each host. The result is put in the OUTPUT_MON_PERF file
and in the iCAT catalog.

The digest of the load factor can be retrieved by the iquest query:
    iquest "SELECT SLD_RESC_NAME,SLD_LOAD_FACTOR"

Example Usage:
acServerMonPerf {
    # This microservice invokes a command in iRODS/server/bin/cmd
    # irodsServerMonPerf - a perl script to get monitoring information
    delay("<PLUSET>30s</PLUSET>< EF>1h</EF>") {
        msiServerMonPerf("default","default");
    }
}
INPUT null
OUTPUT ruleExecOut
4.105   Core :: Framework Services System :: msiSetBulkPutPostProcPolicy

msiSetBulkPutPostProcPolicy ( msParam_t * xflag )

Parameters:
[in]   xflag   - Required - a msParam of type STR_MS_T.
           "on" - enable execution of acPostProcForPut.
           "off" - disable execution of acPostProcForPut.

Description:
This micro-service sets whether the post processing "put" rule (acPostProcForPut) should be run (on or off) for the bulk put operation. Setting the policy to "off" improves performance, as no post processing is done when uploading using the bulk option.

Note:
The policy is implemented in the core.re file.

Example Usage:

acBulkPutPostProcPolicy { msiSetBulkPutPostProcPolicy("off"); }
4.106  Core :: Framework Services System :: msiSetDataObjAvoidResc

msiSetDataObjAvoidResc ( msParam_t * xavoidResc )

Parameters:
[in]  xavoidResc  - a msParam of type STR_MS_T - the name of the resource to avoid

Description:
This micro-service specifies the resource to avoid when opening a file. The copy stored in the specified resource will not be picked unless it is the only copy.

Note:
The policy is implemented in the core.re file.

Example Usage:

acPreProcForDataObjOpen {msiSetDataObjAvoidResc("demoResc");}
msiSetDataObjPreferredResc (msParam_t * xpreferredRescList)

Parameters:
[in] xpreferredRescList - a msParam of type STR_MS_T, comma-delimited list of resources

Description:
If the data object has multiple copies, this micro-service specifies the preferred resource for the opened object.

Note:
The copy stored in this preferred resource will be picked if it exists. More than one resource can be input using the character "%" as separator. e.g., resc1%resc2%resc3. The most preferred resource should be at the beginning of the list.

Example Usage:
acPreProcForDataObjOpen {msiSetDataObjPreferredResc("demoResc%testResc");}
Core :: Framework Services System :: msiSetDataTypeFromExt

**msiSetDataTypeFromExt**

Parameters:
None

Description:
This micro-service checks if the filename has an extension (string following a period (.) and if so, checks if the iCAT has a matching entry for it, and if so sets the dataObj data_type.

Note:
Always returns success since it is only doing an attempt; that is, failure is common and not really a failure.

The types of data recognized by iRODS are:

<table>
<thead>
<tr>
<th>AIX DLL</th>
<th>DICOM image</th>
<th>Mac Executable</th>
<th>SGI DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX Executable</td>
<td>directory shadow object</td>
<td>Mac OSX Executable</td>
<td>SGI Executable</td>
</tr>
<tr>
<td>ascii compressed Huffman</td>
<td>DLL</td>
<td>Movie</td>
<td>SGML File</td>
</tr>
<tr>
<td>ascii compressed Lempel-Ziv</td>
<td>Document</td>
<td>MP3 - MPEG Audio</td>
<td>shadow object</td>
</tr>
<tr>
<td>ascii text</td>
<td>DVI format</td>
<td>MPEG</td>
<td>Slide</td>
</tr>
<tr>
<td>audio streams</td>
<td>ebcdic compressed Huffman</td>
<td>MPEG 3 Movie</td>
<td>Solaris DLL</td>
</tr>
<tr>
<td>AVI</td>
<td>ebcdic compressed Lempel-Ziv</td>
<td>MPEG Movie</td>
<td>Solaris Executable</td>
</tr>
<tr>
<td>binary file</td>
<td>ebcdic text</td>
<td>MSWord Document</td>
<td>Spread Sheet</td>
</tr>
<tr>
<td>BMP -Bit Map</td>
<td>email</td>
<td>NSF Award Abstracts</td>
<td>SQL script</td>
</tr>
<tr>
<td>C code</td>
<td>Excel Spread Sheet</td>
<td>NT DLL</td>
<td>streams</td>
</tr>
<tr>
<td>C include file</td>
<td>Executable</td>
<td>NT Executable</td>
<td>tar bundle</td>
</tr>
<tr>
<td>compressed file</td>
<td>fig image</td>
<td>object code</td>
<td>tar file</td>
</tr>
<tr>
<td>compressed mmCIF file</td>
<td>FITS image</td>
<td>orb data</td>
<td>tcl script</td>
</tr>
<tr>
<td>compressed PDB file</td>
<td>fortran code</td>
<td>pbm image</td>
<td>text</td>
</tr>
<tr>
<td>compressed tar file</td>
<td>generic</td>
<td>PDF Document</td>
<td>tiff image</td>
</tr>
<tr>
<td>Cray DLL</td>
<td>gif image</td>
<td>perl script</td>
<td>Troff format</td>
</tr>
<tr>
<td>Cray Executable</td>
<td>html</td>
<td>PNG-Portable Network Graphics</td>
<td>URL</td>
</tr>
<tr>
<td>CSS-Cascading Style Sheet</td>
<td>image</td>
<td>Postscript format</td>
<td>uuencoded tiff</td>
</tr>
<tr>
<td>data file</td>
<td>java code</td>
<td>Power Point Slide</td>
<td>video streams</td>
</tr>
<tr>
<td>database</td>
<td>jpeg image</td>
<td>print-format</td>
<td>Wave Audio</td>
</tr>
<tr>
<td>database object</td>
<td>LaTeX format</td>
<td>program code</td>
<td>WMV-Windows Media Video</td>
</tr>
<tr>
<td>database shadow object</td>
<td>library code</td>
<td>Quicktime Movie</td>
<td>Word format</td>
</tr>
<tr>
<td>datascope data</td>
<td>link code</td>
<td>realAudio</td>
<td>xml</td>
</tr>
<tr>
<td>DICOM header</td>
<td>Mac DLL</td>
<td>realVideo</td>
<td>XML Schema</td>
</tr>
</tbody>
</table>

Example Usage:

```
acPostProcForPut {msiSetDataTypeFromExt;}
```
4.109 Core :: Framework Services System :: msiSetDefaultResc

**msiSetDefaultResc**

```
msiSetDefaultResc ( msParam_t* xdefaultRescList,
                    msParam_t* xoptionStr )
```

**Parameters:**

- **xdefaultRescList**
  - Required - a msParam of type STR_MS_T which is a list of %delimited resource Names. It is the resource that is used if no resource is input. A "null" means there is no defaultResc.

- **xoptionStr**
  - a msParam of type STR_MS_T which is an option (preferred, forced, null) with null as default. A "forced" input means the defaultResc will be used regardless of the user input. The forced action only applies to users with normal privilege.

**Description:**

This micro-service specifies the resource to use if no resource is input.

**Note:**

This function is no longer mandatory after version 2.3. A "null" means there is no default resource. More than one resource can be input using the character "%" as separator. If it is used, it should be executed right after the screening function msiSetNoDirectRescInp.

**Example Usage:**

```c
acSetRescSchemeForCreate {
    msiSetNoDirectRescInp("testResc");
    msiSetDefaultResc("demoResc","preferred");
    msiSetRescSortScheme("default");
}
```
4.110 Core :: Framework Services System :: msiSetGraftPathScheme

msiSetGraftPathScheme (msParam_t * xaddUserName,
msParam_t * xtrimDirCnt)

Parameters:
[in] xaddUserName - This msParam specifies whether the userName should be added to the physical path. e.g. $vaultPath/$userName/$logicalPath. "xaddUserName" can have two values - yes or no.
[in] xtrimDirCnt - This msParam specifies the number of leading directory elements of the logical path to trim. Sometimes it may not be desirable to graft the entire logical path. e.g., for a logicalPath /myZone/home/me/foo/bar, it may be desirable to graft just the part "home/me/foo/bar" to the vaultPath. "xtrimDirCnt" should be set to 1 in this case. The default value is 1.

Description:
This micro-service sets the VaultPath scheme to GRAFT_PATH. It grafts (adds) the logical path to the vault path of the resource when generating the physical path for a data object.

Note:
The policy is implemented in the core.re file. The default is addUserName == yes and trimDirCnt == 1. If trimDirCnt is greater than 1, the home or trash entry will be taken out.

Example Usage:
acSetVaultPathPolicy {msiSetGraftPathScheme("no","1");}
4.111 Core :: Framework Services System :: msiSetMultiReplPerResc

msiSetMultiReplPerResc( )

Parameters:
None.

Description:
By default, the system allows one copy per resource. This micro-service sets the number of copies per resource to unlimited.

Note:
When multiple replicas are enabled on the same resource, the way the physical file name is specified is modified. If the first copy is stored on:
    /Vault/home/rods/sub1
the second copy will be stored on
    /Vault/replica/home/rods/sub1

Example Usage:

acSetMultiReplPerResc {msiSetMultiReplPerResc;}
4.112  Core :: Framework Services System :: msiSetNoDirectRescInp

\textbf{msiSetNoDirectRescInp} ( msParam_t * xrescList, )

**Parameters:**
\[ \text{[in]} \quad \text{xrescList} \quad - \text{InpParam is a xrescList of type STR_MS_T which is a list of %-delimited resource names e.g., resc1%resc2%resc3.} \]

**Description:**
This micro-service sets a list of resources that cannot be used by a normal user directly. It checks a given list of taboo-resources against the user provided resource name and disallows if the resource is in the list of taboo-resources.

**Note:**
This micro-service is optional, but if used, should be the first function to execute because it screens the resource input.

**Session Variables Used:**
\begin{itemize}
  \item rei->doinp->condInput - user set resource list
  \item rei->rsComm->proxyUser.authInfo.authFlag
\end{itemize}

**Example Usage:**

```
acSetRescSchemeForCreate {
  msiSetNoDirectRescInp("testResc");
  msiSetDefaultResc("demoResc","random");
  msiSetRescSortScheme("byRescType");
}
```
4.113  Core :: Framework Services System :: msiSetNumThreads

**msiSetNumThreads**  (  
msParam_t  *  xsizePerThrInMbStr,  
msParam_t  *  xmaxNumThrStr,  
msParam_t  *  xwindowSizeStr  )

**Parameters:**

[in]  xsizePerThrInMbStr  - 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.

[in]  xmaxNumThrStr  - The maximum number of threads to use. It accepts an integer value up to 16. It also accepts the word "default" which sets maxNumThr to a default value of 4.

[in]  xwindowSizeStr  - 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.

**Description:**

This micro-service specifies the parameters for determining the number of threads to use for data transfer. It sets the number of threads and the TCP window size.

**Note:**

The msiSetNumThreads function must be present or no threads will be used for all transfers. The acSetNumThreads rule supports conditions based on $rescName so that different policies can be set for different resources. For a network bandwidth of 350 MB/sec and a round-trip latency of 100 milliseconds, the xwindowSizeStr should be set to 35 Mbytes to fill the network pipe. If the window size is smaller, multiple I/O streams will be needed.

**Example Usage:**

acSetNumThreads {msiSetNumThreads("32","8","default");}
4.114 Core :: Framework Services System :: msiSetPublicUserOpr

**msiSetPublicUserOpr** (msParam_t * xoprList)

**Parameters:**
- **xoprList** - Only 2 operations are allowed - "read" - read files; "query" – browse some system level metadata. More than one operation can be input using the character "%" as separator. e.g., read%query.

**Description:**
This micro-service sets a list of operations that can be performed by the user "public".

**Note:**
none

**Session Variables Used:**
rei->rsComm->clientUser.authInfo.authFlag

**Example Usage:**
acSetPublicUserPolicy {msiSetPublicUserOpr("read%query");}
4.115 Core :: Framework Services System :: msiSetRandomScheme

msiSetRandomScheme ( )

Parameters:
None

Description:
This micro-service sets the scheme for composing the physical path in the vault to RANDOM. 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.

Note:
none

Example Usage:

acSetVaultPathPolicy {msiSetRandomScheme;}
4.116 Core :: Framework Services System :: msiSetRescQuotaPolicy

msiSetRescQuotaPolicy ( msParam_t * xflag )

Parameters:
[in] xflag - Required - a msParam of type STR_MS_T.
   "on" - enable Resource Quota enforcement
   "off" - disable Resource Quota enforcement (default)

Description:
This micro-service sets the resource quota to on or off.

Note:
none

Example Usage:

acRescQuotaPolicy {msiSetRescQuotaPolicy("on");}
4.117  Core :: Framework Services System :: msiSetRescSortScheme

msiSetRescSortScheme ( msParam_t * xsortScheme )

Parameters:
[in] xsortScheme - 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. This requires that the resource monitoring system 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.

Description:
This micro-service sets the scheme for selecting the best resource to use when creating a data object.

Note:
none

Example Usage:

acSetRescSchemeForCreate {
    msiSetDefaultResc("demoResc","null");
    msiSetRescSortScheme("random");
    msiSetRescSortScheme("byRescClass");
    }

4.118  Core :: Framework Services System :: msiSetReServerNumProc

**msiSetReServerNumProc** (msParam_t * xnumProc)

**Parameters:**

- **[in]**  xnumProc - a STR_MS_T representing number of processes.
  This value can be "default" or an integer

**Description:**

Sets the number of processes to use when running jobs in the irodsReServer. The irodsReServer supports multi-tasking such that one or two long-running jobs will not block the execution of other jobs.

**Note:**

The allowed range is 0-4. A value of 0 means that no forking will be done. The default value is 1.

**Example Usage:**

acSetReServerNumProc {msiSetReServerNumProc("4");}
4.119  Core :: Framework Services System :: msiSetResource

`msiSetResource ( msParam_t * xrescName )`

**Parameters:**

[in]  `xrescName` - is a msParam of type STR_MS_T

**Description:**

This micro-service sets the resource as part of a workflow execution.

**Note:**

None.

**Example Usage:**

```plaintext
acRegisterData {
    ON(ObjPath like "/home/collections.nvo/2mass/fits-images/*") {
        acCheckDataType("fits image");
        msiSetResource("testResc");
        msiRegisterData;
    }
}
```
msiSortDataObj (msParam_t * xsortScheme)

Parameters:
[in] xsortScheme - input sorting scheme

Description:
This micro-service sorts the copies of the data object using a sorting scheme.

Note:
Currently, "random" and "byRescClass" sorting schemes are supported. If "byRescClass" is set, data objects in the "cache" resources will be used ahead of those in the "archive" resources. The sorting schemes can be chained. Thus msiSortDataObj("random"); msiSortDataObj("byRescClass"); means that the data objects will be sorted randomly first and then separated by class.

Example Usage:

acPreProcForDataObjOpen {
    msiSortDataObj("byRescClass");
    msiStageDataObj("demoResc");
}
4.121 Core :: Framework Services System :: msiStageDataObj

msiStageDataObj ( msParam_t * xcacheResc )

Parameters:
[in]  xcacheResc - The resource name in which to cache the object

Description:
This micro-service stages the data object to the specified resource before operation. It stages a copy of the data object in the cacheResc before opening the data object.

Note:
This is typically used to make a copy on a local storage resource. The $writeFlag session variable has been created to be used as a condition for differentiating between open for read ($writeFlag == 0) and open for write ($writeFlag == 1). e.g.:

acPreprocForDataObjOpen {ON($writeFlag == "0") {msiStageDataObj("demoResc");}}
acPreprocForDataObjOpen {ON($writeFlag == "1") { } }
acPreprocForDataObjOpen {
  msiSortDataObj("random");
  msiSetDataObjPreferredResc("xyz%demoResc8%abc");
  msiStageDataObj("demoResc8");
}

Example Usage:

acPreProcForDataObjOpen {
  msiSortDataObj("byRescClass");
  msiStageDataObj("demoResc");
}
4.122 Core :: Framework Services System :: msiSysChksumDataObj

msiSysChksumDataObj ( )

**Parameters:**
None

**Description:**
This micro-service performs a checksum on the just uploaded or copied data object.

**Note:**
The checksum is done at the remote storage location.

**Example Usage:**

acPostProcForPut {msiSysChksumDataObj; }
4.123  Core :: Framework Services System :: msiSysReplDataObj

\texttt{msiSysReplDataObj} \quad (\quad \text{msParam}_t^{*}\quad \text{xcacheResc},\quad \\
\quad \quad \quad \quad \text{msParam}_t^{*}\quad \text{xflag} \quad )

\textbf{Parameters:}
\begin{itemize}
  \item \textbf{[in]} \quad \text{xcacheResc} \quad - \text{storage resource for replica}
  \item \textbf{[in]} \quad \text{xflag} \quad - \text{flag controlling replication}
\end{itemize}
\begin{itemize}
  \item \text{all} \quad – \text{a copy will be made on all resources in a resource group}
  \item \text{null} \quad – \text{only a single copy will be made within the resource group}
  \item \text{updateRepl} \quad – \text{existing stale copies are updated to the latest copy}
  \item \text{rbudpTransfer} \quad – \text{use the Reliable Blast UDP protocol for the transfer}
\end{itemize}

\textbf{Description:}
This micro-service replicates a data object. It can be used to replicate a copy of the just uploaded or copied
data object to the specified replResc.

\textbf{Note:}
The "all" flag is only meaningful if the replResc is a resource group. In this case, setting xflag to "all" means a copy will be made on all of the resources in the resource group. A "null" input means a single copy will be made in one of the resources in the resource group. More than one flag value can be set using the "%" character as a separator. e.g., "all%updateRepl". Here the "all" flag means replicate to all resources in a resource group and update all stale copies since the "updateRepl" flag is also set.

It may be desirable to do replication only if the dataObject is stored in a resource group. For example, the following rule can be used:

\texttt{acPostProcForPut \{ON($rescGroupName \neq \"\") \{msiSysReplDataObj($rescGroupName,"all");\}\}}

\textbf{Example Usage:}

\texttt{acPostProcForPut \{
  \quad \text{ON($rescGroupName \neq \"\") \{
    \quad \text{msiSysReplDataObj($rescGroupName,"all");}
  \}\}
\}}
4.124 Core :: iCAT System Services :: msiAclPolicy

msiAclPolicy ( msParam_t* msParam )

Parameters:
[in] msParam - a msParam of type STR_MS_T – can have value 'STRICT'

Description:
Limits display of information about files owned by a user.

Note:
Should not be used outside of the rules defined in core.re. Once set STRICT, strict mode remains in force
(users cannot call it in another rule to change the mode back to non-strict). See core.re.

If 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 made STRICT by calling msiAclPolicy(STRICT),
the General Query Access Control is applied on collections and data object metadata which means that ils,
etc., will need 'read' access or better to the collection to see a list of the collection contents (name of data-
objects, sub-collections, etc.). Formerly this was controlled at build-time via a GEN_QUERY_AC flag in
config.mk. Default is the normal, non-strict level, allowing users to see other collections. In all cases,
access control to the data-objects is enforced. When "STRICT" is set, the user will only be able to see
their home collection. They will not be able to view the start of the directory path, "/datagrid-name/home",
or "/datagrid-name/trash".

Even with STRICT access control, the admin user is not restricted so various micro-services and queries
will still be able to evaluate system-wide information.

Since iRODS 2.5, $userNameClient is available although this is only secure in a 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. But the typical use is to just set it strict or not for all users.

Example Usage:

    acAclPolicy { msiAclPolicy("STRICT"); }
Core :: iCAT System Services :: msiAddConditionToGenQuery

**msiAddConditionToGenQuery**

```c
msParam_t * attribute,
msParam_t * operator,
msParam_t * value,
msParam_t * queryInput
```

**Parameters:**

- **attribute** - Required - A STR_MS_T with the iCAT attribute name (see www.irods.org/index.php/icatAttributes).
- **operator** - Required - A STR_MS_T with the operator.
- **value** - Required - A STR_MS_T with the value.
- **queryInput** - Required - A GenQueryInp_MS_T.

**Description:**
Adds a condition to a genQueryInp_t structure.

**Note:**
This micro-service adds a condition to an existing genQueryInp_t, based on three parameters. The first is an iCAT attribute index given without its 'COL_' prefix. The second one is the SQL operator. The third one is the value and may contain wildcards. Normally used with msiAddSelectFieldToGenQuery and msiExecGenQuery to build queries from the results of other micro-services or actions within an iRODS rule.

**Example Usage:**

```plaintext
myTestRule {
  # Input parameters are:
  # Attribute name
  # Operator
  # Value
  # Input/Output
  # General query structure
  # Output from running the example is:
  # List of files in collection /tempZone/home/rods

  # initial condition for query corresponds to "COLL_NAME like '/tempZone/home/rods/%%"
  msiMakeGenQuery(*Select,"COLL_NAME like '/tempZone/home/rods/%%",*GenQInp);

  # adding condition to query "DATA_NAME like rule%%"
  msiAddConditionToGenQuery(*Attribute,*Operator,*Value,*GenQInp);
  msiExecGenQuery(*GenQInp,*GenQOut);
  foreach(*GenQOut)
  {
    msiGetValByKey(*GenQOut,"DATA_NAME",*DataFile);
    msiGetValByKey(*GenQOut,"COLL_NAME",*Coll);
    writeLine("stdout","*Coll/*DataFile");
  }
}
```

INPUT *Select="DATA_NAME, COLL_NAME", *Attribute="DATA_NAME", *Operator= like ", *Value="rule%%"
OUTPUT ruleExecOut
4.126 Core :: iCAT System Services :: msiAddSelectFieldToGenQuery

\[\text{msiAddSelectFieldToGenQuery}(\text{select}, \text{function}, \text{queryInput})\]

**Parameters:**
- **select** - Required - A \text{STR_MS_T} with the select field.
- **function** - Optional - A \text{STR_MS_T} with the function. Valid values are [MIN|MAX|SUM|AVG|COUNT]
- **queryInput** - Optional - A \text{GenQueryInp_MS_T} structure.

**Description:**
Sets a select field in a \text{genQueryInp_t} structure.

**Note:**
This micro-service sets a select field in a \text{genQueryInp_t} structure from two parameters. The first is an iCAT attribute index given without its 'COL_' prefix. The second one is the optional SQL operator. A new \text{genQueryInp_t} is created if \text{queryInput} is NULL. The \text{msiAddSelectFieldToGenQuery} micro-service typically follows \text{msiMakeGenQuery} to build and extend queries within a rule.

**Example Usage:**

```plaintext
myTestRule {
    # Input parameters are:
    # Select field
    # Function to apply to attribute
    # Input/Output parameter:
    # GenQuery structure
    # Output from running the example is:
    # List of sizes of collections in /tempZone/home/rods

    # initial select is on COLL_NAME
    msiMakeGenQuery(*Select,"COLL_NAME like '/tempZone/home/rods/%%'",*GenQInp);

    # add select on sum(DATA_SIZE)
    msiAddSelectFieldToGenQuery(*SelectAdd,*Function,*GenQInp);
    msiExecGenQuery(*GenQInp,*GenQOut);
    foreach(*GenQOut)
    {
        msiGetValByKey(*GenQOut,"DATA_SIZE","Size");
        msiGetValByKey(*GenQOut,"COLL_NAME","Coll");
        writeLine("stdout","For collection *Coll, the size of the files is *Size");
    }
}
```

**INPUT** *Select="COLL_NAME", *SelectAdd="DATA_SIZE", *Function="SUM"
**OUTPUT** ruleExecOut
4.127 Core :: iCAT System Services :: msiAddUserToGroup

msiAddUserToGroup ( msParam_t * msParam )

Parameters:
[in] msParam - a msParam of type STR_MS_T, the name of the group

Description:
This micro-service adds a user to a group.

Note:
Should not be used outside of the rules defined in core.re. This is called via an 'iadmin' command.

Example Usage:

acCreateUserF1 {
    # this should only be executed within the core.re file
    msiCreateUser ::: msiRollback;
    acCreateDefaultCollections ::: msiRollback;
    msiAddUserToGroup("public") ::: msiRollback;
    msiCommit;
}
4.128 Core :: iCAT System Services :: msiCommit

msiCommit ( )

Parameters:
None

Description:
This micro-service commits pending database transactions, registering the new state information into the iCAT.

Note:
This is used to commit changes (if any) into the iCAT database as part of a rule and micro-service chain. See core.re for examples. In other cases, iCAT updates and inserts are automatically committed into the iCAT Database as part of the normal operations (in the 'C' code).

Example Usage:

acCreateUserF1 {
  # This is the acCreateUserF1 policy in the core.re file
  ON ($otherUserName == "anonymous")
  {
    msiCreateUser ::: msiRollback;
    msiCommit;
  }
}
4.129 Core :: iCAT System Services :: msiCreateCollByAdmin

```c
msiCreateCollByAdmin (msParam_t * xparColl,
                        msParam_t * xchildName)
```

**Parameters:**
- **[in] xparColl** - a msParam of type STR_MS_T for parent collection
- **[in] xchildName** - a msParam of type STR_MS_T for child collection

**Description:**
This micro-service creates a collection by an administrator executed command.

**Note:**
Should not be used outside of the rules defined in core.re.
This is called via an 'iadmin' command, and can only be executed by a person with a rodsadmin role.

**Example Usage:**
```c
acCreateCollByAdmin(*parColl,*childColl) {
    msiCreateCollByAdmin(*parColl,*childColl);
}
```
4.130  Core :: iCAT System Services :: msiCreateUser

msiCreateUser ( )

Parameters:
None

Description:
This micro-service creates a new user.

Note:
Should not be used outside of the rules defined in core.re. This is called via an 'iadmin' command.

Example Usage:

acCreateUserF1 {
# This is the acCreateUserF1 policy in the core.re file
  msiCreateUser :: msiRollback;
  acCreateDefaultCollections :: msiRollback;
  msiAddUserToGroup("public") :: msiRollback;
  msiCommit;
}

4.131  Core :: iCAT System Services :: msiDeleteCollByAdmin

msiDeleteCollByAdmin (msParam_t * xparColl,
                     msParam_t * xchildName)

Parameters:
[in] xparColl  - a msParam of type STR_MS_T for the parent collection
[in] xchildName  - a msParam of type STR_MS_T for the child collection

Description:
This micro-service deletes a collection by an administrator executed command.

Note:
Should not be used outside of the rules defined in core.re. This is called via an 'iadmin' command. This micro-service can only be executed by a person with a rodsadmin role.

Session Variables Used:
rei->rsComm->clientUser.authFlag (must be admin)

Example Usage:
acDeleteCollByAdmin(*parColl,*childColl) {
    msiDeleteCollByAdmin(*parColl,*childColl);
}
msiDeleteUnusedAVUs ( )

Parameters:
None

Description:
This function deletes unused AVUs from the iCAT. See 'iadmin rum'. This requires execution by a person with a rodsadmin role.

Note:
This causes the unused AVUs to be removed from the ICAT.

Session Variables Used:
rei->rsComm->clientUser.authFlag (must be admin)

Example Usage:

```plaintext
myTestRule {
    delay (*arg1) {
        msiDeleteUnusedAVUs;
    }
}

INPUT *arg1="<PLUSET>1m</PLUSET><EF>24h</EF>"
OUTPUT ruleExecOut
```
4.133 Core :: iCAT System Services :: msiDeleteUser

msiDeleteUser ( )

Parameters:
None

Description:
This micro-service deletes a user.

Note:
Should not be used outside of the rules defined in core.re. This is called via an 'iadmin' command by a person with a rodsadmin role.

When a user is deleted, decisions should also be made about deletion of the data, the files they may have in trash, and the files they may have in bundle and replica directories.

Session Variables Used:
rei->rsComm->clientUser.authFlag (must be admin)

Example Usage:

```chunk
acDeleteUserF1 {
    # This is the acDeleteUserF1 policy in the core.re file
    acDeleteDefaultCollections ::: msiRollback;
    msiDeleteUser ::: msiRollback;
    msiCommit;
}
```
4.134 Core :: iCAT System Services :: msiExecGenQuery

**msiExecGenQuery**

```
(msParam_t * genQueryInParam,
msParam_t * genQueryOutParam)
```

**Parameters:**

- **[in]** genQueryInParam - a msParam of type GenQueryInp_MS_T structure holding the query
- **[out]** genQueryOutParam - a msParam of type GenQueryOut_MS_T structure holding the result

**Description:**
This function executes a given general query structure and returns the first 256 rows of the result.

**Note:**
Takes a SQL-like iRODS query (no FROM clause) and returns a table structure. Use a loop over msiGetMoreRows to get all rows. The example loops over all files in a collection for arbitrarily large collections by working with the continuation index. The processing is done in sets of 256 records at a time. To return more row values, modify MAX_SQL_ROWS in iRODS/ lib/core/include/rodsGenQuery.h.

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  # Structure holding the query
  # Output parameter is:
  # Structure holding the query result
  # Output from running the example is:
  # List of the number of files and size of files in collection /tempZone/home/rods/large-coll
  *ContInxOld = 1;
  *Count = 0;
  *Size = 0;
  msiMakeGenQuery("DATA_ID, DATA_SIZE",*Condition,*GenQInp);
  msiExecGenQuery(*GenQInp, *GenQOut);
  msiGetContInxFromGenQueryOut(*GenQOut,*ContInxNew);
  while(*ContInxOld > 0) {
    foreach(*GenQOut) {
      msiGetValByKey(*GenQOut, "DATA_SIZE", *Fsize);
      *Size = *Size + double(*Fsize);
      *Count = *Count + 1;
    }
    *ContInxOld = *ContInxNew;
    if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp,*GenQOut,*ContInxNew);} }
  writeLine("stdout","Number of files in *Coll is *Count and total size is *Size"); }
INPUT *Coll = "/tempZone/home/rods/large-coll", *Condition="COLL_NAME like "+Coll"
OUTPUT ruleExecOut
```
4.135 Core :: iCAT System Services :: msiExecStrCondQuery

**msiExecStrCondQuery** (msParam_t * queryParam, msParam_t * genQueryOutParam)

**Parameters:**
- [in] queryParam - a msParam of type GenQueryInp_MS_T
- [out] genQueryOutParam - a msParam of type GenQueryOut_MS_T

**Description:**
This function takes a given query string, creates an iCAT query, executes it, and returns the values. This example returns up to 256 rows from the query. To get more results, iterate over msiGetMoreRows as in the example for rulesiExecGenQuery.r.

**Note:**
The query string can also be generated by msiMakeQuery.

**Example Usage:**

```plaintext
myTestRule {
  # Input parameters are:
  # String with conditional query
  # Output parameter is:
  # Result string
  msiExecStrCondQuery(*Select,*QOut);
  foreach(*QOut) {
    msiPrintKeyValPair("stdout",*QOut)
  }
}

INPUT *Select="SELECT DATA_NAME where DATA_NAME like 'rule%%%"
OUTPUT ruleExecOut
```
4.136   Core :: iCAT System Services :: msiExecStrCondQueryWithOptions

**msiExecStrCondQueryWithOptions**

- `msParam_t * queryParam`,
- `msParam_t * zeroResultsIsOK`,
- `msParam_t * maxReturnedRowsParam`,
- `msParam_t * genQueryOutParam`

**Parameters:**

- **queryParam**
  - a msParam of type GenQueryInp_MS_T which holds the query
- **zeroResultsIsOK**
  - Optional - a msParam of type STR_MS_T - must equal "zeroOK". When set, a query that results in no found rows is permissible.
- **maxReturnedRowsParam**
  - Optional - a msParam of type STR_MS_T - as an integer
- **genQueryOutParam**
  - a msParam of type GenQueryOut_MS_T

**Description:**
This function takes a given condition string and options, creates an iCAT query, executes it and returns the values.

**Note:**
This micro-service is deprecated. Starting with iRODS version 3.0, queries that do not return any rows are permissible. This example returns up to 256 rows from the query. To get more results, iterate over msiGetMoreRows as in the example for rulesmsiExecGenQuery.r.

**Example Use**

```plaintext
myTestRule {
    # Input parameters are:
    # String containing conditional query
    # Optional flag to allow null return
    # Optional maximum number of rows returned
    # zeroOK
    # Output parameter is:
    # Results from executing the query
    msiMakeQuery(*Sel,*Cond,*QIn);
    msiExecStrCondQueryWithOptions(*QIn,"zeroOK","15",*QOut);
    foreach(*QOut) {msiPrintKeyValPair("stdout",*QOut);}

    # see if the result is the same from an alternate microservice
    writeLine("stdout","Compare output with msiExecStrCondQuery results");
    msiExecStrQuery(*QIn,*QOut);
    *Count = 0;
    foreach(*QOut) {
        *Count = *Count + 1;
        if(*Count <= 15) {msiPrintKeyValPair("stdout",*QOut);}
    }
}
```

**INPUT** *Sel="DATA_NAME", *Cond="DATA_NAME like 'rule%%""

**OUTPUT** ruleExecOut
4.137  Core :: iCAT System Services :: msiGetContInxFromGenQueryOut

**msiGetContInxFromGenQueryOut**

```
(   msParam_t *   genQueryOutParam,
    msParam_t *   continueInx)
```

**Parameters:**

- **genQueryOutParam**: [in] - Required - of type GenQueryOut_MS_T which holds the query result.
- **continueInx**: [out] - a INT_MS_T containing the new continuation index. A value greater than 1 indicates additional rows are available.

**Description:**

This micro-service gets the continuation index value from genQueryOut generated by msiExecGenQuery.

**Note:**

The output result `continueInx` can be used to determine whether there are remaining rows to retrieve from the generated query. The example loops over queries to the iCAT catalog to get additional rows. The micro-services within the `foreach` loop are executed for each row returned from the iCAT query.

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  # Structure holding the query
  # Output parameter is:
  # Continuation index, non-zero when additional rows are available
  # Output from running the example is:
  # List of the number of files and size of files in collection /tempZone/home/rods
  *ContInxOld = 1;
  *Count = 0;
  *Size = 0;
  msiMakeGenQuery("DATA_ID, DATA_SIZE", *Condition,*GenQInp);
  msiExecGenQuery(*GenQInp, *GenQOut);
  msiGetContInxFromGenQueryOut(*GenQOut,*ContInxNew);
  while(*ContInxOld > 0) {
    if(*ContInxNew == 0) { *ContInxOld = 0; }
    foreach(*GenQOut) {
      msiGetValByKey(*GenQOut, "DATA_SIZE", *Fsize);
      *Size = *Size + double(*Fsize);
      *Count = *Count + 1;
    }
    if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp,*GenQOut,*ContInxNew);}
  }
  writeLine("stdout","Number of files in *Coll is *Count and total size is *Size");
}
```

**INPUT**

```
*Coll = "/tempZone/home/rods/\%\%", *Condition="COLL_NAME like *Coll"
```

**OUTPUT**

```
ruleExecOut
```
4.138 Core :: iCAT System Services :: msiGetMoreRows

```c
msiGetMoreRows ( msParam_t *genQueryInp_msp,
      msParam_t *genQueryOut_msp,
      msParam_t *continueInx )
```

**Parameters:**
[in] `genQueryInp_msp` - Required - a GenQueryInp_MS_T containing the query parameters and conditions.
[in] `genQueryOut_msp` - Required - a GenQueryOut_MS_T to write results to. If its continuation index is 0 the query will be closed.
[out] `continueInx` - a INT_MS_T containing the new continuation index (after the query).

**Description:**
This micro-service continues an unfinished query by returning the next set of 256 rows.

**Note:**
This micro-service gets the next batch of rows for an open iCAT query. This is used after initial msiMakeGenQuery and msiExecGenQuery micro-service calls that have more than 256 rows in the response.

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  # Structure holding the query
  # Structure holding the query result
  # Output parameter is:
  # Continuation index, greater than zero is additional rows can be retrieved
  # Output from running the example is:
  # List of the number of files and size of files in collection /tempZone/home/rods
    *ContInxOld = 1;
    *Count = 0;
    *Size = 0;
    msiMakeGenQuery("DATA_ID, DATA_SIZE", *Condition, *GenQInp);
    msiExecGenQuery(*GenQInp, *GenQOut);
    msiGetContInxFromGenQueryOut(*GenQOut,*ContInxNew);
    while(*ContInxOld > 0) {
      foreach(*GenQOut) {
        msiGetValByKey(*GenQOut, "DATA_SIZE", *Fsize);
        *Size = *Size + double(*Fsize);
        *Count = *Count + 1;
      }
      *ContInxOld = *ContInxNew;
    }
    if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp,*GenQOut,*ContInxNew);}
  writeLine("stdout","Number of files in *Coll is *Count and total size is *Size");
}
INPUT *Coll = "/tempZone/home/rods/%%", *Condition="COLL_NAME like "*Coll"
OUTPUT ruleExecOut
```
4.139 Core :: iCAT System Services :: msiMakeGenQuery

msiMakeGenQuery (msParam_t * selectListStr,
                  msParam_t * condStr,
                  msParam_t * genQueryInpParam)

Parameters:
[in] selectListStr - Required - a STR_MS_T containing the parameters.
[in] condStr - Required - a STR_MS_T containing the conditions
[out] genQueryInpParam - a GenQueryInp_MS_T containing the parameters and conditions.

Description:
This micro-service constructs an SQL string that can be issued to the iCAT catalog by a subsequent call to msiExecGenQuery. The SQL string is contained in a GenQueryInp_MS_T structure.

Note:
This micro-service sets up a genQueryInp_t data structure needed by calls to rsGenQuery(). It is typically executed before calls to msiExecGenQuery and msiGetMoreRows.

Example Usage:

myTestRule {
  # Input parameters are:
  # Selected attribute list
  # Condition for selecting files
  # Output parameter is:
  # Structure holding the query
  # Output from running the example is:
  # List of the number of files and size of files in collection /tempZone/home/rods
  *ContInxOld = 1;
  *Count = 0;
  *Size = 0;
  msiMakeGenQuery("DATA_ID, DATA_SIZE", *Condition, *GenQInp);
  msiExecGenQuery(*GenQInp, *GenQOut);
  msiGetContInxFromGenQueryOut(*GenQOut, *ContInxNew);
  while(*ContInxOld > 0) {
    if(*ContInxNew == 0) { *ContInxOld = 0; }
    foreach(*GenQOut) {
      msiGetValByKey(*GenQOut, "DATA_SIZE", *Fsize);
      *Size = *Size + double(*Fsize);
      *Count = *Count + 1;
    }
    if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp, *GenQOut, *ContInxNew);}
  }
  *ContInxOld = 0;
  *Count = 0;
  *Size = 0;
}

writeLine("stdout","Number of files in *Coll is *Count and total size is *Size");

INPUT *Coll = "/tempZone/home/rods/\*\*", *Condition="COLL_NAME like *Coll"
OUTPUT ruleExecOut
4.140 Core :: iCAT System Services :: msiMakeQuery

```c
msiMakeQuery ( msParam_t * selectListParam,
                msParam_t * conditionsParam,
                msParam_t * queryOutParam )
```

**Parameters:**
- **[in]** selectListParam - a STR_MS_T containing the parameters that are selected in the query.
- **[in]** conditionsParam - a STR_MS_T containing the condition for the query.
- **[out]** queryOutParam - a STR_MS_T containing the parameters and conditions as sql.

**Description:**
Creates an sql query from a parameter list and condition.

**Note:**
This micro-service creates a sql query string from the input parameter list (select statement) and condition.

**Example Usage:**

```c
myTestRule { 
    # Input parameters are:
    # Attribute list
    # Condition for selecting files
    # Output parameter is:
    # SQL execution string
    # Output from running the example is:
    # List of all files that start with rule
    msiMakeQuery(*Select,*Condition,*Query);
    msiExecStrCondQuery(*Query,*GenQOut);
    foreach(*GenQOut) {msiPrintKeyValPair("stdout",*GenQOut);} 
}
```

INPUT *Select="DATA_NAME, COLL_NAME, DATA_RESC_NAME, DATA_REPL_NUM, DATA_SIZE", *Condition="DATA_NAME like 'rule%%'"
OUTPUT ruleExecOut
4.141 Core :: iCAT System Services :: msiPrintGenQueryInp

msiPrintGenQueryInp ( msParam_t * where,  
                       msParam_t * genQueryInpParam )

Parameters:
[in] where - Required - a STR_MS_T specifying the output buffer.
[in] genQueryInpParam - Required - a GenQueryInp_MS_T containing the parameters and conditions.

Description:
This micro-service prints the given GenQueryInp_MS_T structure to the given target buffer. This provides a way to list the contents of a query that is being issued to the iCAT metadata catalog.

Note:
The target buffer can be "stdout", "stderr", "serverLog", or an internal buffer.

Example Usage:

myTestRule {
  # Input parameter is:
  # Buffer where string will be written
  # GenQueryInp string
  # Output from running the example is:
  # Selected Column 501 With Option 1
  # Selected Column 407 With Option 4
  # Condition Column 501 like '/tempZone/home/rods/%%'

  msiMakeGenQuery(*Select,"COLL_NAME like '/tempZone/home/rods/%%'",*GenQInp);

  # add select on sum(DATA_SIZE)
  msiAddSelectFieldToGenQuery(*SelectAdd,*Function,*GenQInp);
  msiPrintGenQueryInp("stdout",*GenQInp);
}

INPUT *Select="COLL_NAME", *SelectAdd="DATA_SIZE",*Function="SUM"
OUTPUT *GenQInp, ruleExecOut
4.142 Core :: iCAT System Services :: msiPrintGenQueryOutToBuffer

**msiPrintGenQueryOutToBuffer**

```c
(msParam_t * queryOut,
msParam_t * format,
msParam_t * buffer)
```

**Parameters:**
- **[in]** `queryOut` - Required - A GenQueryOut_MS_T structure holding the query result.
- **[in]** `format` - Optional - A STR_MS_T with a C-style format string, similar to the iquest icommand format.
- **[out]** `buffer` - A BUF_LEN_MS_T structure for the result.

**Description:**
Writes the contents of the output results from a query contained in a GenQueryOut_MS_T structure into a buffer.

**Note:**
The results can be formatted with an optional C-style format string the same way it is done in iquest. The format string specifies how the selected attributes will be printed. The format string:

```
" %5.5s access has been given to user %6.6s for the file %s"
```

will map the attributes in the SELECT statement to the "%" variables in the format string

```
SELECT DATA_ACCESS_NAME, USER_NAME, DATA_NAME
```

in the order they are listed. Thus DATA_ACCESS_NAME replaces the first "%" and is listed as a string of 5 characters, USER_NAME replaces the second "%" and is listed as a string of 6 characters, and DATA_NAME replaces the third "%" and is listed as a string or arbitrary length.

**Example Usage:**

```c
myTestRule {

# Input parameters are:
#  GenQueryOut structure
#  C-style format string
# Output parameter is:
#  Buffer for result

  msiMakeGenQuery("DATA_ID, DATA_SIZE",*Condition,*GenQInp);
  msiExecGenQuery(*GenQInp, *GenQOut);
  msiPrintGenQueryOutToBuffer(*GenQOut,*Form,*Buf);
  writeBytesBuf("stdout",*Buf);
}

INPUT *Coll = "/tempZone/home/rods/%%", *Condition="COLL_NAME like "*Coll"", *Form="For data-
ID %6.6s the data size is %8.8s"
OUTPUT ruleExecOut
```
4.143  Core :: iCAT System Services :: msiQuota

msiQuota  (     )

Parameters:
None

Description:
Calculates storage usage and checks quota values (over/under/how-much-used).

Note:
Causes the ICAT quota tables to be updated. This must be executed by a person with a rodsadmin role.

Session Variables Used:
rei->rsComm->clientUser.authFlag (must be admin)

Example Usage:

myTestRule {
    # Administrator command to cause update to iCAT quota tables
    delay("<PLUSET>30s</PLUSET><EF>24h</EF>") {
        msiQuota;
        writeLine("serverLog","Updated quota check");
    }
}

INPUT null
OUTPUT ruleExecOut
4.144  Core :: iCAT System Services :: msiSetQuota

msiQuota( msParam_t *type,
         msParam_t *name,
         msParam_t *resource,
         msParam_t *value )

Parameters:
[in] type - a STR_MS_T - Can be either "user" or "group"
[in] name - a STR_MS_T with the name of the user or group
[in] resource - Optional - a STR_MS_T with the name of the resource where the
               quota will apply, or "total" for the quota to be system-wide.
[in] value - an INT_MST_T or DOUBLE_MS_T or STR_MS_T with the
             quota value (in bytes)

Description:
This micro-service sets a storage quota for a given user or group of users for either a specific storage
system or for total storage.

Note:
If no resource name is provided, the quota will apply across all resources. The micro-service requires
rodsadmin privileges.

Example Usage:

myTestRule {  
  # Input parameters are:  
  # Type of quota (user or group)  
  # User or group name  
  # Optional resource on which the quota applies (or total for all resources)  
  # Quota value in bytes  
  msiSetQuota(*Type, *Name, *Resource, *Value);  
  writeLine("stdout","Set quota on *Name for resource *Resource to *Value bytes");  
}  
INPUT *Type="user", *Name="rods", *Resource="demoResc", *Value="1000000000"  
OUTPUT ruleExecOut
### msiRenameCollection

**Parameters:**
- `oldName` - a `msParam` of type `STR_MS_T` with the old collection name.
- `newName` - a `msParam` of type `STR_MS_T` with the new name for the collection.

**Description:**
This function renames a collection and is used via a rule with `msiRenameLocalZone`.

**Note:**
Should not be used outside of the rules defined in `core.re`. This is called via an 'iadmin' command.

**Example Usage:**

```c
acRenameLocalZone(*oldZone,*newZone) {
    msiRenameCollection("/"+str(*oldZone)++"",*newZone) ::: msiRollback;
    msiRenameLocalZone(*oldZone,*newZone) ::: msiRollback;
    msiCommit;
}
```
4.146   Core :: iCAT System Services :: msiRenameLocalZone

msiRenameLocalZone (  msParam_t *   oldName,
                      msParam_t *   newName        )

Parameters:
[in]   oldName   -  a msParam of type STR_MS_T
[in]   newName   -  a msParam of type STR_MS_T

Description:
This micro-service renames the LOCALZone by updating multiple tables in iCAT. This can only be
executed by a person with the rodsadmin role.

Note:
Should not be used outside of the rules defined in core.re. This is called via an 'iadmin' command.

Example Usage:

acRenameLocalZone(*oldZone,*newZone){
    msiRenameCollection(""++ str(*oldZone) ++ ","*,newZone) ::: msiRollback;
    msiRenameLocalZone(*oldZone,*newZone) ::: msiRollback;
    msiCommit;
}
4.147 Core :: iCAT System Services :: msiRollback

**msiRollback** ( )

**Parameters:**
None

**Description:**
This function deletes user and collection information from the iCAT by rolling back the database transaction.

**Note:**
This is used to reverse changes to the iCAT database as part of a rule and micro-service recovery chain. See core.re for examples. In other cases, iCAT updates and inserts are automatically rolled-back as part of the normal operations (from within the 'C' code of a micro-service).

**Example Usage:**

```c
acRenameLocalZone(*oldZone,*newZone) {
    msiRenameCollection("" ++ str(*oldZone) ++ "",*newZone) ::: msiRollback;
    msiRenameLocalZone(*oldZone,*newZone) ::: msiRollback;
    msiCommit;
}
```
Core :: iCAT System Services :: msiServerBackup

msiServerBackup ( msParam_t *options,
                 msParam_t *keyValOut )

Parameters:
[in] options  - Optional - a STR_MS_T that contains one or more options in the format
keyWd1=value1+++keyWd2=value2+++keyWd3=value3...
Currently no options have been defined. This is a placeholder for future options.
[out] keyValOut - a KeyValPair_MS_T with the number of files and bytes written.

Description:
Copies iRODS server files to the local resource

Note:
Copies server files to the local vault and registers them into iCAT. Object (.o) files and binaries are not included.

Example Usage:

myTestRule {
# Input parameter is:
# Options - currently none are specified for controlling server backup
# Output parameter is:
# Result - a keyvalpair structure holding number of files and size
# This will take a while to run.
# Backup files are stored in a directory as hostname_timestamp:
# $ ils system_backups
# /tempZone/home/rods/system_backups:
# C- /tempZone/home/rods/system_backups/localhost_2011-08-19.16:00:29
#
# msiServerBackup(*Opt,*Result);
# writeKeyValPairs("stdout",*Result, " : ");
}

INPUT *Opt=""
OUTPUT ruleExecOut
4.149  Core :: iCAT System Services :: msiSetACL

msiSetACL( msParam_t * recursiveFlag,
         msParam_t * accessLevel,
         msParam_t * userName,
         msParam_t * pathname )

Parameters:
[in] recursiveFlag  - a STR_MS_T, either "default" or "recursive", "recursive" is only relevant if set with accessLevel set to "inherit".
[in] accessLevel    - a STR_MS_T containing one of the following permissions:
                       null
                       read
                       write
                       own
                       inherit
[in] userName       - a STR_MS_T, the user name or group name who will have ACL changed. For user names in a federated data grid, use user_name#zone_name
[in] pathName       - a STR_MS_T, the collection or data object that will have its ACL changed.

Description:
This micro-service changes the ACL for a given pathname, either a collection or a data object.

Note:
For collections, the modification can be recursive and the inheritance bit can be changed as well. The list of access controls is arranged from lowest level to highest level. When an access control is set, all lower level access controls are also enabled. Only a single access control is saved per person. The new access permissions are only those set by the application of the micro-service.

Example Usage:

myTestRule { 
  # 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
  msiSetACL("default",*Acl,*User,*Path);
  writeLine("stdout","Set owner access for *User on file *Path");
}

INPUT *User="testuser", *Path="/tempZone/home/rodsub1/foo3", *Acl = "write"
OUTPUT ruleExecOut
4.150 Core :: iCAT System Services :: msiVacuum

msiVacuum ( )

Parameters:
None

Description:
Postgres vacuum, done periodically to optimize indices and performance.

Note:
The effect of this is that the iCAT database gets optimized. This micro-service works with PostgreSQL only.

Example Usage:

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


4.151  Core :: Email Micro-services :: msiSendMail

**msiSendMail** (msParam_t* xtoAddr, 
msParam_t* xsSubjectLine, 
msParam_t* xbody)

**Parameters:**
- [in] xtoAddr - a msParam of type STR_MS_T which is an address of the receiver.
- [in] xsSubjectLine - a msParam of type STR_MS_T which is a subject of the message.
- [in] xbody - a msParam of type STR_MS_T which is a body of the message.

**Description:**
Sends email.

**Note:**
This micro-service sends e-mail using the mail command in the unix system. The first argument is the e-mail address of the receiver. The second argument is the subject string and the third argument is the body of the e-mail. No attachments are supported. The sender of the e-mail is the unix userid running the irodsServer.

**Example Usage:**

```plaintext
myTestRule {
  # Input parameters are:
  #  Address
  #  Subject of e-mail
  #  Message body
  msiSendMail(*Address,*Subject,*Body);
  writeLine("stdout","Sent e-mail to *Address about *Subject");
}
INPUT *Address="irod-chat@googlegroups.com",*Subject="Test message",*Body="Testing the msiSendMail microservice"
OUTPUT ruleExecOut
```
4.152   Core :: Email Micro-services :: msiSendStdoutAsEmail

msiSendStdoutAsEmail (  msParam_t * xtoAddr,
                        msParam_t * xssubjectLine)

Parameters:
[in] xtoAddr    - a msParam of type STR_MS_T which is the address of the receiver.
[in] xssubjectLine - a msParam of type STR_MS_T which is the subject of the message.

Description:
Sends the current buffer content in rei->ruleExecOut->stdoutBuf.buf as email.

Note:
This micro-service, given a xtoAddr parameter (an e-mail address) and a xssubjectLine parameter, sends out
the stdout buffer as the body of the e-mail.

Example Usage:

myTestRule {
  # Input parameters are:
  # Address
  # Subject
  writeLine("stdout","Message from stdout buffer");
  msiSendStdoutAsEmail(*Address,*Subject);
  writeLine("stdout","Sent e-mail to *Address about *Subject");
}
INPUT *Address="irod-chat@googlegroups.com", *Subject="Test message"
OUTPUT ruleExecOut
4.153 Core :: Key-Value (Attr-Value) :: msiAddKeyVal

msiAddKeyVal ( msParam_t * inKeyValPair, 
               msParam_t * key, 
               msParam_t * value )

Parameters:
[in, out] inKeyValPair - Optional - a KeyValPair_MS_T
[in] key - Required - A STR_MS_T containing the key
[in] value - Optional - A STR_MS_T containing the value

Description:
Adds a new key and value to a keyValPair_t structure.

Note:
A new keyValPair_t structure is created if inKeyValPair is NULL. The example uses a micro-service from the ERA module, which must be enabled and compiled.

Example Usage:

myTestRule {
    # Input parameters are:
    # Key-value buffer (may be empty)
    # Key
    # Value
    msiGetSystemTime(*Time,"human");
    msiAddKeyVal(*Keyval,*Key,*Time);
    msiAssociateKeyValuePairsToObj(*Keyval,*Coll,-"C");
    msiGetCollectionPSmeta(*Coll,*Buf);
    writeBytesBuf("stdout",*Buf);
}
INPUT *Coll="/tempZone/home/rods/sub1", *Key="TimeStamp"
OUTPUT ruleExecOut
4.154  Core :: Key-Value (Attr-Value) :: msiAssociateKeyValuePairsToObj

msiAssociateKeyValuePairsToObj (  msParam_t *  metadataParam,
                                 msParam_t *  objParam,
                                 msParam_t *  typeParam )

Parameters:
[in] metadataParam  - a msParam of type KeyValPair_MS_T holding the key-value structure.
[in] objParam       - a msParam of type STR_MS_T that specifies the object to which the metadata will be added.
[in] typeParam      - a msParam of type STR_MS_T that defines the type of object.

Description:
This micro-service associates <key,value> pairs from a given KeyValPair_t structure with an object.

Note:
The object type is also needed:
   - d for data object
   - R for resource
   - G for resource group
   - C for collection
   - u for user

The example uses a micro-service from the ERA module, which must be enabled and compiled.

Example Usage:

myTestRule {  
  # Input parameters are:
  # Key-value buffer (may be empty)
  # Key
  # Value
  msiGetSystemTime(*Time,"human");
  msiAddKeyVal(*Keyval,"TimeStamp","Time");
  msiAssociateKeyValuePairsToObj(*Keyval,*Coll,"-C");
  msiGetCollectionPSmeta(*Coll,*Buf);
  writeBytesBuf("stdout",*Buf);
}

INPUT *Coll="/tempZone/home/rods/sub1"
OUTPUT ruleExecOut
4.155 Core :: Key-Value (Attr-Value) :: msiGetValByKey

msiGetValByKey ( msParam_t * inKVPair,
                 msParam_t * inKey,
                 msParam_t * outVal )

Parameters:
[in] inKVPair - This msParam is of type KeyValPair_PI which is a KeyValPair List.
inKey - This msParam is of type STR_MS_T which is a key.
[out] outVal - This msParam is of type STR_MS_T which is a value corresponding to key.

Description:
Given a list of KVPairs and a Key, this micro-service gets the corresponding value.

Note:
This is used to extract metadata from a query into a variable for use by subsequent micro-services.

Example Usage:

myTestRule {
    # Input parameters are:
    # Key-value pair list
    # Key
    # Output parameter is:
    # Value
    # Output from running the example is:
    # List of file in the collection
    writeLine("stdout","List files in collection *Coll");
    msiExecStrCondQuery("SELECT DATA_NAME where COLL_NAME = '*Coll',*QOut);
    foreach (*QOut) {
        msiGetValByKey(*QOut,"DATA_NAME",*File);
        writeLine("stdout","*File");
    }
}

INPUT *Coll="/tempZone/home/rods/sub1"
OUTPUT ruleExecOut
4.156 Core :: Key-Value (Attr-Value) :: msiPrintKeyValPair

```c
msiPrintKeyValPair ( msParam_t * where,
                     msParam_t * inkvpair )
```

**Parameters:**
- **[in]** `where` - a msParam of type STR_MS_T which is either stderr or stdout.
- **[in]** `inkvpair` - a msParam of type KeyValPair_PI which is a KeyValPair list (structure).

**Description:**
Prints out a row of the key-value structure to the stdout buffer.

**Note:**
It takes a row-structure from GenQueryOut_MS_T and prints it as a ColumnName=Value pair. The rule uses the result (tabular structure) from execution of an iCAT query. In the example, the micro-service msiExecStrCondQuery is used to run the query: `SELECT DATA_TYPE_NAME WHERE COLL_NAME = */tempZone/home/rods/sub1"`. The result is printed using the msiPrintKeyValPair micro-service, which prints each row as an attribute-value pair. A separator line is printed after each row.

**Example Usage:**

```c
myTestRule {
    # Input parameters are:
    #  Location where data is written
    #    stdout
    #    stderr
    #  Structure holding key-value pairs
    # Example lists metadata for an input file path
    msiSplitPath(*Path,*Coll,*File);
    msiExecStrCondQuery("SELECT *Meta where COLL_NAME = *Coll' and DATA_NAME = *File",*QOut);
    foreach(*QOut) {msiPrintKeyValPair("stdout",*QOut);}
}
```

Input: *Path="/tempZone/home/rods/sub1/", *Meta="DATA_TYPE_NAME"

Output: ruleExecOut
4.157  Core :: Key-Value (Attr-Value) :: msiRemoveKeyValuePairsFromObj

msiRemoveKeyValuePairsFromObj ( msParam_t * metadataParam,
                         msParam_t * objParam,
                         msParam_t * typeParam )

Parameters:
[in]  metadataParam  - a msParam of type KeyValPair_MS_T with the attributes to be removed
[in]  objParam       - a msParam of type STR_MS_T defining the object from which the attributes
                        will be removed
[in]  typeParam      - a msParam of type STR_MS_T defining the type of object

Description:
This micro-service removes <key,value> pairs from an iRODS object.

Note:
The object type is also needed:
- d for data object
- R for resource
- G for resource group
- C for collection
- u for user

The example uses a micro-service from the ERA module, which will need to be enabled and compiled.

Example Usage:

myTestRule {  
# Input parameters are:
#  Key-value pair list
#  Path to object
#  Type of object (-d, -C)
# Output from running the example is:
#  Add metadata

  msiString2KeyValPair(*Str,*Keyval);
  msiAssociateKeyValuePairsToObj(*Keyval,*Path,"-d");

  # List metadata
  writeLine("stdout","List metadata on file");
  msiGetDataObjPSmeta(*Path,*Buf);
  writeBytesBuf("stdout",*Buf);

  # Remove metadata
  msiRemoveKeyValuePairsFromObj(*Keyval,*Path,"-d");

  # List metadata remaining on file
  writeLine("stdout","list metadata after removing *Str");
  msiGetDataObjPSmeta(*Path,*Buf);
  writeBytesBuf("stdout",*Buf);
}

INPUT *Path="/tempZone/home/rods/sub1/foo3", *Str="Testmeta=deletetest"
OUTPUT ruleExecOut
4.158  Core :: Key-Value (Attr-Value) :: msiStrArray2String

```
msiStrArray2String ( msParam_t * inSAParam, 
                    msParam_t * outStr )
```

**Parameters:**

- **[in]** `inSAParam` - a `msParam` of type `strArr_MS_T` which is an array of strings.
- **[out]** `outStr` - a `msParam` of type `STR_MS_T` which is a string with `%`-separators.

**Description:**
An array of strings is converted to a string separated by `%`-signs.

**Note:**
In the example, a string of `%`-separated key-value strings is converted to key-value pairs. The string is also converted to a string array, which is then converted back to a string and printed.

**Example Usage:**

```c
myTestRule { 
   # Input parameter is:
   # Input string - `%`-separated key=value strings
   # Output parameter is:
   # String array buffer
   writeLine("stdout","Input string is *Str");
   msiString2KeyValPair(*Str,*Keyval);
   writeKeyValPairs("stdout",*Keyval," : ");
   msiString2StrArray(*Str,*Stray);
   msiStrArray2String(*Stray, *Str2);
   writeLine("stdout","After conversion to array and back, string is");
   writeLine("stdout",*Str2);
}

INPUT *Str="key1=value1%key2=value2%key3=value3"
OUTPUT ruleExecOut
```
4.159 Core :: Key-Value (Attr-Value) :: msiString2KeyValPair

msiString2KeyValPair (msParam_t * inBufferP,
                     msParam_t * outKeyValPairP)

Parameters:
[in] inBufferP - a msParam of type STR_MS_T which is key=value pairs separated by %-sign.
[out] outKeyValPairP - a msParam of type KeyValPair_MS_T which is a keyValuePair structure.

Description:
This micro-service converts a %-separated key=value pair of strings into a keyValPair structure.

Note:
The example uses a micro-service from the ERA module, which must be enabled and compiled.

Example Usage:

myTestRule {
    # Input parameter is:
    # String with %-separated key=value pair strings
    # Output parameter is:
    # Key-value structure
    writeLine("stdout","Add metadata string *Str to *Path");
    msiString2KeyValPair(*Str,*Keyval);
    msiAssociateKeyValuePairsToObj(*Keyval,*Path,"-d");
    msiGetDataObjPSmeta(*Path,*Buf);
    writeBytesBuf("stdout",*Buf);
}

INPUT *Str="Tester=rods%Event=document", *Path="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
4.160  Core :: Key-Value (Attr-Value) :: msiString2StrArray

msiString2StrArray (msParam_t * inStr, msParam_t * outSAParam)

Parameters:
[in]  inStr       - a msParam of type STR_MS_T which a string with % -separators.
[out] outSAParam  - a msParam of type strArr_MS_T which is an array of strings.

Description:
A string separated by % -signs is converted to a string array.

Note:
In the example, a string is converted to a string array, and then converted back to a % -separated string and printed.

Example Usage:

myTestRule {
  # Input parameter is:
  # Input string - % -separated key=value strings
  # Output parameter is:
  # String array buffer
  writeLine("stdout","Input string is *Str");
  msiString2KeyValPair(*Str,*Keyval);
  writeKeyValPairs("stdout",*Keyval,":");
  msiString2StrArray(*Str,*Stray);
  msiStrArray2String(*Stray,*Str2);
  writeLine("stdout","After conversion to array and back, string is");
  writeLine("stdout",*Str2);
}

INPUT *Str="key1=value1%key2=value2%key3=value3"
OUTPUT ruleExecOut
4.161 Core :: Key-Value (Attr-Value) :: writeKeyValPairs

writeKeyValPairs (msParam_t * where, msParam_t * inKVPair, msParam_t * separator)

Parameters:
[in] where - a msParam of type STR_MS_T which is the buffer name in ruleExecOut. It can be stdout or stderr.
[in] inKVPair - a msParam of type KeyValPair_MS_T
[in] separator - Optional - a msParam of type STR_MS_T, the desired parameter

Description:
This micro-service writes keyword value pairs to stdout or stderr, using the given separator.

Note:
The writeLine micro-service treats the "%" sign as a comment, and does not print the end of the input string after the "%" sign. The rest of the rule works correctly. The example uses a micro-service from the ERA module, which must be enabled and compiled.

Example Usage:

myTestRule {
# Input parameters are:
# String with %-separated key=value pair strings
# Output parameter is:
# Key-value structure
writeLine("stdout","Add metadata string *Str to *Path");
msiString2KeyValPair(*Str,*Keyval);
writeKeyValPairs("stdout",*Keyval,*Status);
msiAssociateKeyValuePairsToObj(*Keyval,*Path,-d);
msiGetObjectPSmeta(*Path,*Buf);
writeBytesBuf("stdout",*Buf);
}

INPUT *Str="Tester=rods%Event=document", *Path="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
4.162 Core :: Other User :: msiApplyDCMetadataTemplate

`msiApplyDCMetadataTemplate (   msParam_t *   inpParam,
                                msParam_t *   outParam   )`

**Parameters:**
- [in] `inpParam` - a STR_MS_T containing the target object's path
- [out] `outParam` - an INT_MS_T containing the status

**Description:**
This micro-service adds 15 empty Dublin Core Metadata fields to an object or collection.

**Note:**
The Dublin Core Metadata fields are:
  - attribute: DC.Creator
  - attribute: DC.Format
  - attribute: DC.Date
  - attribute: DC.Source
  - attribute: DC.Identifier
  - attribute: DC.Type
  - attribute: DC.Contributor
  - attribute: DC.Publisher
  - attribute: DC.Description
  - attribute: DC.Relation
  - attribute: DC.Subject
  - attribute: DC.Coverage
  - attribute: DC.Rights
  - attribute: DC.Title
  - attribute: DC.Language

The example uses a micro-service from the ERA module, which must be enabled and compiled.

**Example Usage:**

```c
myTestRule {
  # Input parameter is:
  # Path of object
  # Output parameter is:
  # Status
  msiApplyDCMetadataTemplate(*Path,*Status);
  msiGetDataObjPSmeta(*Path,*Buf);
  writeBytesBuf("stdout",*Buf);
}
INPUT *Path="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
```
4.163 Core :: Other User :: msiExtractNaraMetadata

msiExtractNaraMetadata ( )

Parameters:
None

Description:
This micro-service is deprecated. It is replaced with msiLoadMetadataFromDataObj in the ERA module. This micro-service extracts NARA style metadata from a configuration file that should be stored in iRODS/server/config/reConfigs/naraMetadata.txt.

Note:
The micro-service is executed as an action within the core.re file. In the example, metadata extraction is automated as part of acPostProcForPut. When a file is put into iRODS, the file iRODS/server/config/reConfigs/naraMetadata.txt will be examined. If the name of the file corresponds to a name listed in naraMetadata.txt, the associated metadata attribute name and attribute value will be added to the iRODS data object. The format of naraMetadata.txt is:
File-path|attribute-name|attribute-value

Example Usage:

acPostProcForPut {msiExtractNaraMetadata;}

4.164 Core :: Other User :: msiExtractTemplateMDFromBuf

```c
msiExtractTemplateMDFromBuf( msParam_t * bufParam,
                              msParam_t * tagParam,
                              msParam_t * metadataParam )
```

**Parameters:**
- **[in]** `bufParam` - a msParam of type BUF_MS_T
- **[in]** `tagParam` - a msParam of type TagStruct_MS_T
- **[out]** `metadataParam` - a msParam of type KeyValPair_MS_T

**Description:**
This micro-service uses a template to parse a buffer containing metadata and create a Key-Value Pairs structure.

**Note:**
The template defines triplets <pre-string-regexp, keyword, post-string-regexp>. The triplets are read into memory, and used to search a metadata buffer. For each set of pre and post regular expressions, the string between them is associated with the specified keyword. All <key, value> pairs found are stored in a KeyValPair_t structure.

In the example, the tag file has the format:
```
<PRETAG>X-Mailer: </PRETAG>Mailer User</POSTTAG>
</POSTTAG>
<PRETAG>Date: </PRETAG>Sent Date</POSTTAG>
</POSTTAG>
<PRETAG>From: </PRETAG>Sender</POSTTAG>
</POSTTAG>
<PRETAG>To: </PRETAG>Primary Recipient</POSTTAG>
</POSTTAG>
<PRETAG>Cc: </PRETAG>Other Recipient</POSTTAG>
</POSTTAG>
<PRETAG>Subject: </PRETAG>Subject</POSTTAG>
</POSTTAG>
<PRETAG>Content-Type: </PRETAG>Content Type</POSTTAG>
</POSTTAG>
```
The end tag is actually a "return" for unix systems, or a "carriage-return/line-feed" for Windows systems.

**Example Usage:**
```
myTestRule {   
  # Input parameters are:
  # Buffer
  # Tag structure
  # Output parameter is:
  # Keyval pair buffer

  #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="/tempZone/home/rods/test/email.tag", *Pathfile="/tempZone/home/rods/test/sample.email",
*Outfile="/tempZone/home/rods/test/sample.email", *Len=10000
OUTPUT ruleExecOut
4.165  Core :: Other User :: msiFreeBuffer

**msiFreeBuffer** (msParam_t * memoryParam)

**Parameters:**
[in] memoryParam - the buffer to free

**Description:**
This micro-service frees a named buffer, including stdout and stderr

**Note:**
Can be used to free a buffer that was previously allocated.

**Example Usage:**

myTestRule {
   # Input parameter is:
   # Buffer to free (can be variable buffer or stdout or stderr)
   msiDataObjOpen(*Flags,*F_desc);
   msiDataObjRead(*F_desc,*Len,*Buf);
   msiDataObjClose(*F_desc,*Status);
   msiFreeBuffer(*Buf);
   writeLine("stdout","Freed buffer");
}

INPUT *Flags="objPath=/tempZone/home/rods/sub1/foo1", *Len="100"
OUTPUT ruleExecOut
4.166 Core :: Other User :: msiGetDiffTime

msiGetDiffTime (msParam_t * inpParam1,
msParam_t * inpParam2,
msParam_t * inpParam3,
msParam_t * outParam )

Parameters:
[in] inpParam1 - a STR_MS_T containing the start date (system time in seconds)
in[ ] inpParam2 - a STR_MS_T containing the end date (system time in seconds)
in[ ] inpParam3 - Optional - a STR_MS_T containing the desired output format (human)
[out] outParam - a STR_MS_T containing the time elapsed between the two dates

Description:
This micro-service returns the difference between two system times

Note:
The default output format is in seconds. Use "human" as the third input parameter for human readable format that converts to days, hours, minutes, and seconds.

Example Usage:

myTestRule {
  # Input parameters are:
  # Start date in system time in seconds
  # End date in system time in seconds
  # Optional format (human)
  # Output parameter is:
  # Duration
  msiGetICatTime(*Start,"unix");
  msiSleep("10",""");
  msiGetICatTime(*End,"unix");
  writeLine("stdout","Start time is *Start");
  msiGetDiffTime(*Start,*End,"",*Dur);
  writeLine("stdout","End time is *End");
  writeLine("stdout","Duration is *Dur");
}

INPUT null
OUTPUT ruleExecOut
4.167 Core :: Other User :: msiGetIcatTime

msiGetIcatTime ( msParam_t * timeOutParam,
msParam_t * typeInParam)

Parameters:
[ out ] timeOutParam - a msParam of type STR_MS_T with the system time
[ in  ] typeInParam - a msParam of type STR_MS_T for type of output
    "icat" or "unix" will return seconds since epoch
    otherwise, human friendly

Description:
This function returns the system time for the ICAT server

Note:
This function returns the system time for the ICAT server in either seconds since the epoch, or in a format
that specifies year-month-day.hour:minute:second.

Example Usage:

myTestRule {
    # Input parameters are:
    #  Time type (icat/unix or human) in seconds
    # Output parameter is:
    #  Time value
    msiGetIcatTime(*Start,"unix");
    msiGetIcatTime(*End,"human");
    writeLine("stdout","Time in seconds is *Start");
    writeLine("stdout","Time human readable is *End");
}
INPUT null
OUTPUT ruleExecOut
**4.168 Core :: Other User :: msiGetSystemTime**

```c
msiGetSystemTime ( msParam_t * outParam,
                    msParam_t * inpParam )
```

**Parameters:**
- `[out]` `outParam` - a STR_MS_T containing the time
- `[in]` `inpParam` - Optional - a STR_MS_T containing the desired output format (human)

**Description:**
This micro-service returns the local system time of the iRODS server.

**Note:**
Default output format is system time in seconds, use "human" as input parameter for human readable format in year-month-day.hour:minute:second.

**Example Usage:**

```c
myTestRule { 
    # Input parameters are:
    # Time type "icat" or "unix" returns time in seconds
    # "human" returns date in Year-Month-Day.Hour:Minute:Second
    # Output parameter is:
    # Time value for local system
    msiGetSystemTime(*Start,"unix");
    msiGetSystemTime(*End,"human");
    writeLine("stdout","Time in seconds is *Start");
    writeLine("stdout","Time human readable is *End");
}
```

**INPUT** null
**OUTPUT** ruleExecOut
4.169 Core :: Other User :: msiGetTaggedValueFromString

msiGetTaggedValueFromString ( 
    msParam_t * inTagParam,  
    msParam_t * inStrParam,  
    msParam_t * outValueParam  )

Parameters:

[in] inTagParam - a msParam of type STR_MS_T
[in] inStrParam - a msParam of type STR_MS_T
[out] outValueParam - a msParam of type INT_MS_T

Description:

This micro-service gets a tagged value from a string. When given a tag-name, this micro-service gets the value from a file in tagged-format (pseudo-XML).

Note:

This performs some regular expression matching. Given a regular expression as a tag-value "t", it identifies the corresponding string in the match string with a string that matches a sub-string value: "<t>.*</t>". The service is used for processing a tagged structure. In this example, "IP-address" is successfully parsed from the tagged string:

<table>
<thead>
<tr>
<th>in</th>
<th>Tag</th>
<th>Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>String</td>
<td>&lt;Mail&gt;IP-address&lt;/Mail&gt;</td>
</tr>
<tr>
<td>out</td>
<td>Value</td>
<td>IP-address</td>
</tr>
</tbody>
</table>

Example Usage:

myTestRule {
    # Input parameters are:
    # Tag string
    # Input string
    # Output parameter is:
    # Value associated with tag
    writeLine("stdout","String that is tested is");
    writeLine("stdout","*Str");
    msiGetTaggedValueFromString(*Tag,*Str,*Val);
    writeLine("stdout","Found value is *Val");
}

INPUT *Tag="Mail", *Str="<Mail>IP-address</Mail>"
OUTPUT ruleExecOut
4.170  Core :: Other User :: msiHumanToSystemTime

msiHumanToSystemTime  (   msParam_t*  inpParam,
                          msParam_t*  outParam   )

Parameters:
[in]   inpParam    - a STR_MS_T containing the input date
[out]  outParam    - a STR_MS_T containing the timestamp

Description:
Converts a human readable date to a system timestamp.

Note:
Expects an input date in the form: YYYY-MM-DD.hh:mm:ss

Example Usage:

myTestRule {  
# Input parameter is:
# Date in human readable form
# Output parameter is:
# Time stamp in seconds since epoch
  msiGetSystemTime(*Date,"human");
  msiHumanToSystemTime(*Date,*Time);
  WriteLine("stdout","Input date is *Date");
  WriteLine("stdout","Time in unix seconds is *Time");
}

INPUT null
OUTPUT ruleExecOut
Core :: Other User :: msiReadMDTemplateIntoTagStruct

**msiReadMDTemplateIntoTagStruct**

Parameter types:
- `msParam_t *` `bufParam`
- `msParam_t *` `tagParam`

**Parameters:**
- [in] `bufParam` - a msParam of type BUF_LEN_MS_T
- [out] `tagParam` - a return msParam of type TagStruct_MS_T

**Description:**
This micro-service parses a buffer containing a template-style file and stores the tags in a tag structure.

**Note:**
The template buffer should contain triplets of the form:

<PRETAG>re1</PRTAG>kw<POSTAG>re2</POSTAG>

"re1" identifies the pre-string. "re2" identifies the post-string and any value between re1 and re2 in a metadata buffer will be associated with the keyword "kw".

**Example Usage:**

```c
myTestRule {
    # Input parameter is:
    # Tag buffer
    # Output parameter is:
    # Tag structure

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

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

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

    # Write result to stdout
    writeKeyValPairs("stdout",*Keyval," : ");

    # Add metadata to the file
    msiGetType(*Outfile,*Otype);
    msiAssociateKeyValuePairsToObj(*Keyval,*Outfile,*Otype);
}
```

INPUT *Tag="/tempZone/home/rods/test/email.tag", *Pathfile="/tempZone/home/rods/test/sample.email", *Outfile="/tempZone/home/rods/test/sample.email", *Len=10000
OUTPUT ruleExecOut
4.172 Core :: Other User :: msiRegisterData

msiRegisterData(
)

Parameters:
None

Description:
Register a new data object into the iRODS data grid.

Note:
Use this only within a core.re file as data object information has to be set in the rei structure.

Example Usage:

acRegisterData { msiRegisterData ::: misRollback; }
4.173  Core :: Other User :: msiStrToBytesBuf

msiStrToBytesBuf (  msParam_t * str_msp,
                  msParam_t * buf_msp )

Parameters:
[in] str_msp   - a STR_MS_T with the input string
[out] buf_msp - a BUF_LEN_MS_T with the string converted to binary.

Description:
Converts a string to a bytesBuf_t structure for use within micro-services.

Note:
The example converts a string to a bytes-buffer, then writes the buffer into a file stored within iRODS.

Example Usage:

myTestRule {  
# Input parameters are:
#  String
#  Output parameter is:
#  Buffer

# Convert string to bytes buffer
msiStrToBytesBuf(*Str,*Buf);

# Create a file and write buffer into the file
msiDataObjCreate(*Path,*Flags,*F_desc);
msiDataObjWrite(*F_desc,*Buf,*Len);
msiDataObjClose(*F_desc,*Status);

# Write the string to stdout
writeLine("stdout","Wrote *Str into file *Path");
}

INPUT *Str="Test string for writing into a file", *Path="/tempZone/home/rods/sub1/foo2",
*Flags="forceFlag="
OUTPUT ruleExecOut
4.174  Core :: Other User :: writeBytesBuf

writeBytesBuf  (  msParam_t * where,
               msParam_t * inBuf   )

Parameters:
[in] where - a msParam of type STR_MS_T which is the buffer name in ruleExecOut. It can be "stdout" or "stderr".
[in] inBuf  - a msParam of type STR_MS_T - related to the status output

Description:
This micro-service writes the buffer in an inOutStruct to stdout or stderr.

Note:
none

Example Usage:

myTestRule {
  # Input Parameters are:
  # Location for write (stdout, stderr)
  # String buffer

  # Make a query
  msiMakeGenQuery("DATA_ID, DATA_SIZE",*Condition,*GenQInp);

  # Issue the query and retrieve query result
  msiExecGenQuery(*GenQInp, *GenQOut);

  # Convert result to a buffer
  msiPrintGenQueryOutToBuffer(*GenQOut,*Form,*Buf);

  # write the result buffer
  writeBytesBuf("stdout",*Buf);
}

INPUT *Coll="/tempZone/home/rods/sub1", *Condition="COLL_NAME like "+coll"", *Form="For data-
ID %6.6s the data size is %8.8s"
OUTPUT ruleExecOut
4.175  Core :: Other User :: writePosInt

writePosInt  ( msParam_t * where,
               msParam_t * inInt )

Parameters:
[in]  where - a msParam of type STR_MS_T which is the buffer name in ruleExecOut.
[in]  inInt  - the integer to write

Description:
This micro-service writes a positive integer into a buffer.

Note:
To add an end of line, write a null after the positive integer using writeLine.

Example Usage:

myTestRule {  
  # Input parameters are:
  # Location (stdout, stderr)
  # Integer
  *A = 1;
  writeLine("stdout","Wrote an integer");
  writePosInt("stdout",*A);
  writeLine("stdout","");
}
INPUT null
OUTPUT ruleExecOut
Core :: Rule-oriented Database Access :: msiRdaCommit

**msiRdaCommit** ( )

**Parameters:**
None

**Description:**
Deprecated micro-service. Use msiDbrCommit instead.

This micro-service calls Remote DataBase Access functions to interface to an arbitrary database (under iRODS access control), performing a commit operation.

**Note:**
This is replaced with the new Database Resources capability. The RDA system requires enabling

```
#define BUILD_RDA 1
```

in

```
iRODS/server/re/scr/reRDA.c
iRODS/server/icat/src/rdaHighLevelRoutines.c
```

and running `gmake`. Then edit `iRODS/server/config/rda.config` to add

```
RDA-name database-user-name password
RDA rdaDbUser 456
```

Edit the `/pgsql/etc/odbc.ini` file if using `postgresql`, and add

```
[iRODS_RDA] Driver=/scratch/slocal/rods/pg5/pgsql/lib/libodbcpsql.so
Debug=0
CommLog=0
Servername=zuri.sdsc.edu
ReadOnly=no
Port=5432
Database=RDA_DB
```

Grant access to the RDA to specific users. This requires rodsadmin privileges.

**Example Usage:**

```plaintext
myTestRule {
# Input parameter:
# None
  msiRdaNoResults(*rda, *sql, "null", "null", "null", "null") ::: msiRdaRollback;
  msiRdaCommit;
}
INPUT *rda="RDA", *sql="create table t2 (c1 varchar(20), c2 varchar(20))"
OUTPUT ruleExecOut
```
4.177 Core :: Rule-oriented Database Access :: msiRdaNoResults

```c
msiRdaNoResults ( msParam_t * inpRdaName,
               msParam_t * inpSQL,
               msParam_t * inpParam1,
               msParam_t * inpParam2,
               msParam_t * inpParam3,
               msParam_t * inpParam4 )
```

**Parameters:**
- `[in]` `inpRdaName` - a STR_MS_T, name of the RDA being used
- `[in]` `inpSQL` - a STR_MS_T which is the SQL
- `[in]` `inpParam1` - Optional - STR_MS_T parameters (bind variables) for the SQL.
- `[in]` `inpParam2` - Optional - STR_MS_T parameters (bind variables) for the SQL.
- `[in]` `inpParam3` - Optional - STR_MS_T parameters (bind variables) for the SQL.
- `[in]` `inpParam4` - Optional - STR_MS_T parameters (bind variables) for the SQL.

**Description:**
This micro-service is deprecated. Use msiDbrExec instead. This micro-service calls Remote Database Access functions to interface to an arbitrary database (under iRODS access control), performing an SQL command that does not return results.

**Note:**
This is replaced with the new Database Resources capability.
This requires irods admin.

**Example Usage:**

```plaintext
myTestRule {
  # Input parameters are:
  # Name of RDA database access file
  # SQL string to be sent to RDA database
  # 3-6 optional bind variables for SQL string
  msiRdaNoResults(*rda, *sql, "null", "null", "null", "null");
  msiRdaCommit;
}
```

INPUT *rda="RDA", *sql="create table t2 (c1 varchar(20), c2 varchar(20))"
OUTPUT ruleExecOut
4.178 Core :: Rule-oriented Database Access :: msiRdaRollback

msiRdaRollback ( )

Parameters:
None

Description:
This micro-service is deprecated. Use msiDbrRollback instead. This micro-service calls RDA functions to interface to an arbitrary database (under iRODS access control), performing a rollback operation.

Note:
This is replaced with the new Database Resources capability. This requires irods admin.

Example Usage:

myTestRule {
  # Input parameter:
  # None
  msiRdaNoResults(*rda, *sql, "null", "null", "null", "null") ::: msiRdaRollback;
  msiRdaCommit;
}

INPUT *rda="RDA", *sql="create table t2 (c1 varchar(20), c2 varchar(20))"
OUTPUT ruleExecOut
4.179  Core :: Rule-oriented Database Access :: msiRdaToDataObj

```c
msiRdaToDataObj (   msParam_t * inpRdaName,
                    msParam_t * inpSQL,
                    msParam_t * inpParam1,
                    msParam_t * inpParam2,
                    msParam_t * inpParam3,
                    msParam_t * inpParam4,
                    msParam_t * inpOutObj )
```

**Parameters:**
- `[in]` `inpRdaName` - a STR_MS_T, name of the RDA being used
- `[in]` `inpSQL` - a STR_MS_T which is the SQL
- `[in]` `inpParam1` - Optional - STR_MS_T parameters (bind variables) for the SQL.
- `[in]` `inpParam2` - Optional - STR_MS_T parameters (bind variables) for the SQL.
- `[in]` `inpParam3` - Optional - STR_MS_T parameters (bind variables) for the SQL.
- `[in]` `inpParam4` - Optional - STR_MS_T parameters (bind variables) for the SQL.
- `[in]` `inpOutObj` - a INT_MS_T, open descriptor to write results to.

**Description:**
This micro-service is deprecated. Use msiDboExec instead. This micro-service calls RDA functions to interface to an arbitrary database (under iRODS access control), getting results (i.e. from a query) and writing them to an open Data Object within iRODS.

**Note:**
This is replaced with the new Database Resources capability. This requires irods admin.

**Example Usage:**

```plaintext
myTestRule { 
  # Input parameters are:
  # Name of RDA database access file
  # SQL string that will be sent to the database
  # 1-4 Optional bind variables for the SQL statement
  # File descriptor into which the results will be written
  msiDataObjCreate(*Path, *Oflag, *D_FD);
  msiRdaToDataObj(*rda,*sql,"null","null","null","null",*D_FD);
  msiDataObjClose(*D_FD,*Status); }
INPUT *rda="RDA", *sql="select * from t2", *Path="/tempZone/home/rods/test/sqlOut.1", *Oflag="forceFlag="
OUTPUT ruleExecOut
```
4.180 Core :: Rule-oriented Database Access :: msiRdaToStdout

```c
msiRdaToStdout ( msParam_t* inpRdaName,
                 msParam_t* inpSQL,
                 msParam_t* inpParam1,
                 msParam_t* inpParam2,
                 msParam_t* inpParam3,
                 msParam_t* inpParam4 )
```

Parameters:
- `inpRdaName` - a STR_MS_T, name of the RDA being used
- `inpSQL` - a STR_MS_T which is the SQL
- `inpParam1` - Optional - STR_MS_T parameters (bind variables) for the SQL.
- `inpParam2` - Optional - STR_MS_T parameters (bind variables) for the SQL.
- `inpParam3` - Optional - STR_MS_T parameters (bind variables) for the SQL.
- `inpParam4` - Optional - STR_MS_T parameters (bind variables) for the SQL.

Description:
This micro-service is deprecated. Use msiDboExec instead. This micro-service calls new RDA functions to interface to an arbitrary database (under iRODS access control), getting results (i.e. from a query) and returning them in stdout.

Note:
This is replaced with the new Database Resources capability.
This requires irods admin.

Example Usage:
```
myTestRule {
  # Input parameters are:
  # Name of RDA database access file
  # SQL string to be sent to RDA database
  # 1-4 Optional bind variables for SQL string
  msiRdaToStdout(*rda,*sql,"null","null","null","null");
}
```

INPUT *rda="RDA", *sql="select * from t2"
OUTPUT ruleExecOut
4.181 Module :: ERA :: msiCollectionSpider

**msiCollectionSpider**

```
(msParam_t * collection,
 msParam_t * objects,
 msParam_t * action,
 msParam_t * status)
```

**Parameters:**
- **collection** - A CollInp_MS_T or a STR_MS_T with the iRODS path
- **objects** - A DataObjInp_MS_T structure used to hold file name
- **action** - A STR_MS_T with the micro-service sequence. The string should be encapsulated in double back-ticks: `\{string\}`
- **status** - An INT_MS_T containing the operation status.

**Description:**
Applies a workflow specified in "action" to all data objects in a collection, recursively.

**Note:**
This micro-service crawls an iRODS collection recursively, and executes a sequence of micro-services/actions for each data object. The data object is passed as a DataObjInp_MS_T structure to the micro-service sequence, through the "objects" parameter. All other parameters in the workflow are internal variables to the workflow. The example uses msiGetObjectPath to convert from the DataObjInp_MS_T structure to a string.

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  # Collection
  # Internal object structure used in workflow
  # Action that will be applied on all files in Path
  # Output parameter:
  # Status
  *Work="`{
    msiDataObjChksum(*File,"ChksumAll",*chk);
    # convert from object structure to a string for printing
    msiGetObjectPath(*File,*obj,*status);
    writeLine("stdout","Checksum for *obj is *chk");
  }``;
  msiCollectionSpider(*Coll,*File,*Work,*Status);
  writeLine("stdout","Operations completed");
}
```

INPUT *Coll="/tempZone/home/rods/sub1"
OUTPUT ruleExecOut
4.182 Module :: ERA :: msiCopyAVUMetadata

```c
msiCopyAVUMetadata (msParam_t * inpParam1,
                     msParam_t * inpParam2,
                     msParam_t * outParam)
```

**Parameters:**
- **[in]** `inpParam1` - A STR_MS_T with the iRODS path of the source object.
- **[in]** `inpParam2` - A STR_MS_T with the iRODS path of the destination object.
- **[out]** `outParam` - An INT_MS_T containing the status.

**Description:**
Copies metadata triplets from one iRODS object to another.

**Note:**
The source and destination can be a data object or a collection, independently.

**Example Usage:**

```c
def myTestRule {
    # Requires acPostProcForModifyAVUMetadata rule in core.re of form
    # acPostProcForModifyAVUMetadata(*Option,*ItemType,*ItemName,*AName,*AValue)|nop|nop
    # Input parameters are:
    #  Source path
    #  Destination path
    # Output parameter is:
    #  Status
    # Output from running the example is:
    #  /tempZone/home/rods/sub1/foo1, State1
    #  Metadata copied from /tempZone/home/rods/sub1/foo1 to /tempZone/home/rods/sub1/foo2
    writeLine("stdout"," *Source, *Flag");
    msiFlagDataObjWithAVU(*Source,*Flag,*Status);
    msiCopyAVUMetadata(*Source,*Dest,*Status);
    writeLine("stdout","Metadata copied from *Source to *Dest");
}
```

**INPUT** `*Source="/tempZone/home/rods/sub1/foo1", *Flag="State1", *Dest="/tempZone/home/rods/sub1/foo2"`

**OUTPUT** `ruleExecOut`
4.183  Module :: ERA :: msiCreateUserAccountsFromDataObj

msiCreateUserAccountsFromDataObj ( msParam_t * inpParam,
                                      msParam_t * outParam )

Parameters:
[in]   inpParam   - a msParam of type DataObjInp_MS_T or STR_MS_T
[out]  outParam   - a msParam of operation status INT_MS_T

Description:
This micro-service creates new user accounts according to information in an iRODS object.

Note:
The file format for user accounts is User-name|User-ID|User-type|Zone| where
    User-name   - iRODS account name without the zone
    User-ID     - not used, set to 001
    User-type   - rosgroup
      rodsadmin
      rodsuser
      domainadmin
      groupadmin
      storageadmin
      rodscurators
    Zone        - zone name

Example Usage:

myTestRule {
  # Input parameters are:
  # Path
  # File format for user accounts is
  #   User-name|User-ID|User-type|Zone|
  #   guest|001|rodsuser|tempZone
  # Output parameter is:
  #   Status
  msiCreateUserAccountsFromDataObj(*Path,*Status);
  writeLine("stdout","Add user accounts defined in file *Path");
}

INPUT *Path="/tempZone/home/rods/testcoll/rodsaccount"
OUTPUT ruleExecOut
4.184 Module :: ERA :: msiDeleteUsersFromDataObj

```c
msiDeleteUsersFromDataObj ( msParam_t * inpParam,
                              msParam_t * outParam )
```

**Parameters:**

- **[in]** `inpParam` - a `msParam` of type `DataObjInp_MS_T` or `STR_MS_T`
- **[out]** `outParam` - a `msParam` of operation status `INT_MS_T`

**Description:**

This micro-service parses an iRODS object for user accounts to delete.

**Note:**

Multiple accounts may be listed, separated by a pipe character.

**Example Usage:**

```plaintext
myTestRule {
    # Input parameter is:
    # Path of file containing accounts to delete
    # Format of file is
    #  user-name|
    #  guest|
    #  assumes the zone is the same as the zone of the client
    # Output parameter is:
    #  Status
    msiDeleteUsersFromDataObj(*Path,*Status);
    writeLine("stdout","User accounts deleted as specified in file *Path");
}
```

```plaintext
INPUT *Path="/tempZone/home/rods/testcoll/rodsaccountdelete"
OUTPUT ruleExecOut
```
4.185  Module :: ERA :: msiExportRecursiveCollMeta

msiExportRecursiveCollMeta ( msParam_t * inpParam, msParam_t * outParam )

Parameters:

[in]  inpParam - A CollInp_MS_T or a STR_MS_T with the iRODS path of the target collection.
[out] outParam - A BUF_LEN_MS_T containing the results.

Description:
Exports metadata AVU triplets for a collection and its contents.

Note:
This micro-service recursively exports user-defined metadata attributes (AVUs) for a collection and all collections and data objects in this collection. The output is written to a bytesBuf_t buffer in pipe-separated format, one line per attribute:

file-name|metadata-name|metadata-value

Example Usage:

myTestRule {
# Input parameter is:
# Path of target collection
# Output parameter is:
# Buffer holding results
# Output from running the example lists:
# file-name|metadata-name|metadata-value
  msiExportRecursiveCollMeta(*Path,*Buf);
  writeBytesBuf("stdout",*Buf);
}

INPUT *Path="/tempZone/home/rods/sub1"
OUTPUT ruleExecOut
4.186 Module :: ERA :: msiFlagDataObjwithAVU

msiFlagDataObjwithAVU ( msParam_t * dataObj,  
                        msParam_t * flag,  
                        msParam_t * status )  

Parameters:
[in] dataObj - A DataObjInp_MS_T or a STR_MS_T with the target object's path.
[in] flag - A STR_MS_T with the flag name.
[out] status - An INT_MS_T containing the operation status.

Description:
Flags a data object with an AVU triplet.

Note:
This micro-service flags a data object by adding to it an AVU. The attribute name is the flag and the value is 1. No unit is specified.

Example Usage:

myTestRule {
# Requires acPostProcForModifyAVUMetadata rule in core.re of form
# acPostProcForModifyAVUMetadata(*Option,*ItemType,*ItemName,*AName,*AValue)||nop|nop  
  # Input parameters are:
  #  Source path
  #  Flag name
  #  Status
  # Output from running the example is:
  # Metadata attribute called State1 is added to /tempZone/home/rods/sub1/foo1
  msiFlagDataObjwithAVU(*Source,*Flag,*Status);
  writeLine("stdout","Metadata attribute called *Flag is added to *Source");
}
INPUT *Source="/tempZone/home/rods/sub1/foo1", *Flag="State1"
OUTPUT ruleExecOut
4.187 Module :: ERA :: msiFlagInfectedObjs

msiFlagInfectedObjs ( msParam_t * scanResObj, msParam_t * scanResc, msParam_t * status )

Parameters:

[in] scanResObj - A DataObjInp_MS_T or a STR_MS_T, which is an iRODS file that contains the results of the virus scan on the specified resource.

[in] scanResc - A STR_MS_T with the name of the resource where the scan took place.

[out] status - An INT_MS_T containing the operation status.

Description:

Parses an output file that was generated by clamscan and flags infected objects with the results of the virus check.

Note:
This requires installation of the clamscan utility in iRODS/server/bin/cmd. The clamscan virus check is assumed to have been run before the msiFlagInfectedObjs micro-service is executed. This micro-service is part of a set of rules and micro-services that constitute an accession policy demonstration for the NARA Transcontinental Persistent Archive.

Example Usage:

myTestRule {
    # Input parameter is:
    # Standard output from clamscan
    # Resource where clamscan was run
    # Output parameter is:
    # Status
    # Execute the clamscan (Clam AntiVirus) utility "clamscan -ri VAULT_DIR"
    # Note that the *VaultPath is the physical path for *Resource on *Host
    # clamscan looks at the physical files in the iRODS vault
    msiExecCmd("scanvault.py", *VaultPath, *Host, "null", "null", *CmdOut);

    # Extract result of the scan of the files in the vault on the specified host
    msiGetStdoutInExecCmdOut(*CmdOut, *StdoutStr);
    msiGetSystemTime(*Time, "human");

    # Write result to an iRODS file
    msiDataObjCreate(*OutputObj ++ "." ++ *Time ++ ".txt", "renci-vault1", *D_FD);
    msiDataObjWrite(*D_FD, *StdoutStr, W_LEN);
    msiDataObjClose(*D_FD, *Status);
    writePosInt("stdout", *Status); writeLine("stdout", ");

    # Execute the routine to extract information from the output file
    msiFlagInfectedObjs(*OutputObj, *Resource, *Status);
}

INPUT *VaultPath="/home/rodsdev/loadingVault/",
*Host="yellow.ils.unc.edu", *OutputObj="/tempZone/home/rods/loading/SCAN_RESULT", *Resource = "loadingResc"
OUTPUT ruleExecOut
4.188 Module :: ERA :: msiGetAuditTrailInfoByActionID

\[
\text{msiGetAuditTrailInfoByActionID} \quad (\text{msParam}_t^* \text{inpParam1}, \text{msParam}_t^* \text{inpParam2}, \text{msParam}_t^* \text{outParam})
\]

**Parameters:**
- [in] \(\text{inpParam1}\) - a msParam of type STR_MS_T for audit action ID
- [in] \(\text{inpParam2}\) - a msParam of type BUF_LEN_MS_T
- [out] \(\text{outParam}\) - a msParam of operation status INT_MS_T

**Description:**
This function gets audit trail information for a specified action identifier.

**Note:**
The action IDs that can be checked are located in file iRODS/server/icat/include/icatDefines.h:

- ACCESS_GRANTED 1000
- REGISTER_DATA_OBJ 2010
- REGISTER_DATA_REPLICA 2011
- UNREGISTER_DATA_OBJ 2012
- REGISTER_DELAYED_RULE 2020
- MODIFY_DELAYED_RULE 2021
- DELETE_DELAYED_RULE 2022
- REGISTER_RESOURCE 2030
- DELETE_RESOURCE 2031
- DELETE_USER_RE 2040
- REGISTER_COLL_BY_ADMIN 2050
- REGISTER_COLL 2051
- DELETE_COLL_BY_ADMIN 2060
- DELETE_COLL 2061
- DELETE_ZONE 2062
- REGISTER_ZONE 2064
- MOD_USER_NAME 2070
- MOD_USER_TYPE 2071
- MOD_USER_ZONE 2072
- MOD_USER_DN 2073 /* no longer used */
- MOD_USER_INFO 2074
- MOD_USER_COMMENT 2075
- MOD_USER_PASSWORD 2076
- ADD_USER_AUTH_NAME 2077
- DELETE_USER_AUTH_NAME 2078
- MOD_GROUP 2080
- MOD_RESR 2090
- MOD_RESR_FREE_SPACE 2091
- MOD_RESR_GROUP 2092
- MOD_ZONE 2093
- REGISTER_USER_RE 2100
- ADD_AVU_METADATA 2110
- DELETE_AVU_METADATA 2111
- COPY_AVU_METADATA 2112
- ADD_AVU_WILD_METADATA 2113
- MOD_ACCESSCONTROL_OBJ 2120
- MOD_ACCESSCONTROL_COLL 2121
- MOD_ACCESSCONTROL_COLL_RECURSIVE 2122
- MOD_ACCESSCONTROL_RESOURCE 2123
- RENAME_DATA_OBJ 2130
**Example Usage:**

```c
myTestRule {
    # Turn on audit trail in iRODS/server/ica/src/icatMidLevelRoutines.c
    # Input parameters are defined in:
    # iRODS/server/ica/include/icatDefines.h
    # Identifier of action
    # 1000 - grant access
    # 2040 - delete user
    # 2061 - delete collection
    # 2076 - modify user password
    # 2120 - modify access control on file
    # Buffer name to be used by microservice
    # Output parameter is:
    # Status
    msiGetAuditTrailInfoByActionID(*Id,*Buf,*Status);
    writeBytesBuf("stdout",*Buf);
}
INPUT *Id="2040"
OUTPUT ruleExecOut
```
4.189 Module :: ERA :: msiGetAuditTrailInfoByKeywords

**msiGetAuditTrailInfoByKeywords**

(msParam_t * inpParam1,
msParam_t * inpParam2,
msParam_t * outParam)

**Parameters:**

- [in] inpParam1 - a msParam of type STR_MS_T for search word within audit comment field.
- [in] inpParam2 - a msParam of type BUF_LEN_MS_T for the result buffer.
- [out] outParam - a msParam of type INT_MS_T for operation status.

**Description:**

This function gets audit trail information by keywords in the comment field.

**Note:**

The comment field is parsed for presence of the specified keyword. Examples of words present in the comment field include action attributes, user account names, etc.

- read
- guest
- delete

**Example Usage:**

```c
myTestRule {
    # Turn on audit trail in iRODS/server/icat/src/icatMidLevelRoutines.c
    # Input parameters are:
    # Keyword to search for within audit comment field
    #   read
    #   delete
    #   guest
    # Buffer to hold result
    # Output parameter is:
    # Status
    msiGetAuditTrailInfoByKeywords(*Keyword,*Buf,*Status);
    writeBytesBuf("stdout",*Buf);
}
INPUT *Keyword="%guest%"
OUTPUT ruleExecOut
```
4.190  Module :: ERA :: msiGetAuditTrailInfoByObjectID

msiGetAuditTrailInfoByObjectID ( msParam_t * inpParam1, msParam_t * inpParam2, msParam_t * outParam )

Parameters:
[in]  inpParam1  - a msParam of type STR_MS_T for Data object ID
[in]  inpParam2  - a msParam of type BUF_LEN_MS_T for result buffer
[out] outParam   - a msParam of operation status INT_MS_T

Description:
This function gets audit trail information for a specified data object identifier.

Note:
Data object identifiers can be found by querying the iCAT catalog for "DATA_ID".

Example Usage:

myTestRule {
    # Input parameters are:
    # Object identifier
    # Buffer for results
    # Output parameter is:
    # Status
    msiSplitPath(*Path, *Coll, *File);
    msiExecStrCondQuery("SELECT DATA_ID where COLL_NAME = '*Coll' and DATA_NAME = '*File'",*QOut);
    foreach(*QOut) {
        msiGetValByKey(*QOut,"DATA_ID",*Objid);
        msiGetAuditTrailInfoByObjectID(*Objid,*Buf,*Status);
        writeBytesBuf("stdout",*Buf);
    }
}

INPUT *Path="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
4.191  Module :: ERA :: msiGetAuditTrailInfoByTimeStamp

**msiGetAuditTrailInfoByTimeStamp**  
(msParam_t * inpParam1,  
msParam_t * inpParam2,  
msParam_t * inpParam3,  
msParam_t * outParam)

**Parameters:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>inpParam1 - a msParam of type STR_MS_T for Start time in Unix seconds</td>
</tr>
<tr>
<td>[in]</td>
<td>inpParam2 - a msParam of type STR_MS_T for Stop time in Unix seconds</td>
</tr>
<tr>
<td>[in]</td>
<td>inpParam3 - a msParam of type BUF_LEN_MS_T for results</td>
</tr>
<tr>
<td>[out]</td>
<td>outParam - a msParam of operation status INT_MS_T</td>
</tr>
</tbody>
</table>

**Description:**
This micro-service gets audit trail information by the timestamp in unix seconds.

**Note:**
The current time in the form year-month-day.hour.minute:second can be converted to unix seconds by calling msiHumanToSystemTime.

**Example Usage:**

```
myTestRule {  
  # Input parameters are:
  # Start time in unix seconds
  # Stop time in unix seconds
  # Buffer for results
  # Output parameter is:
  # Status
  # Output from example is audit trail from Startdate to current time
  msiGetIcatTime(*End,"unix");
  msiHumanToSystemTime(*Startdate,*Start);
  msiGetAuditTrailInfoByTimeStamp(*Start,*End,*Buf,*Status);
  writeBytesBuf("stdout",*Buf);
}
```

INPUT *Startdate="2011-08-30.01:00:00"  
OUTPUT ruleExecOut
4.192 Module :: ERA :: msiGetAuditTrailInfoByUserID

msiGetAuditTrailInfoByUserID ( msParam_t * inpParam1,
                                msParam_t * inpParam2,
                                msParam_t * outParam )

Parameters:
[in]  inpParam1   - a msParam of type STR_MS_T for person ID
[in]  inpParam2   - a msParam of type BUF_LEN_MS_T for result buffer
[out] outParam    - a msParam of operation status INT_MS_T

Description:
This micro-service gets audit trail information for a specified user identifier.

Note:
The user identifier can be retrieved by querying the iCAT catalog for the attribute "USER_ID".

Example Usage:

myTestRule {  
  # Input parameters are:
  # User identifier
  # Buffer for results
  # Output parameter is:
  # Status
  msiExecStrCondQuery("SELECT USER_ID where USER_NAME = '*Person'",*QOut);
  foreach(*QOut) {  
    msiGetValByKey(*QOut,"USER_ID",*PersonID);
    msiGetAuditTrailInfoByUserID(*PersonID,*Buf,*Status);
    writeBytesBuf("stdout",*Buf);
  }
}

INPUT *Person="rods"
OUTPUT ruleExecOut
4.193 Module :: ERA :: msiGetCollectionACL

msiGetCollectionACL ( msParam_t * inpParam1, msParam_t * inpParam2, msParam_t * outParam )

Parameters:
[in] inpParam1 - A CollInp_MS_T or a STR_MS_T with the iRODS path of the target collection.
[in] inpParam2 - Optional - A STR_MS_T. Set it to "recursive" to perform the operation recursively.
[out] outParam - A BUF_LEN_MS_T containing the results.

Description:
Gets ACL (Access Control List) for a collection.

Note:
The access controls are returned in a pipe-separated format:
    File-path|user-name|access

The types of access controls are:
    null     delete metadata
    execute  administer object
    read annotation create object
    read system metadata modify object
    read metadata delete object
    read object create token
    write annotation delete token
    create metadata curate
    modify metadata own

Example Usage:

myTestRule {  
  # Input parameters are:
  # Path of collection  
  # Flag  
  # recursive - to get ACLs for sub-collections and files  
  # Output parameter is:
  # Buffer holding the result  
  msiGetCollectionACL(*Path,"recursive",*Buf);  
  writeBytesBuf("stdout",*Buf);  
}

INPUT *Path="/tempZone/home/rods"
OUTPUT ruleExecOut
4.194  Module :: ERA :: msiGetCollectionContentsReport

msiGetCollectionContentsReport ( msParam_t * inpParam1,
                                msParam_t * outParam1,
                                msParam_t * outParam2 )

Parameters:
[in]  inpParam1   - A CollInp_MS_T or a STR_MS_T with the iRODS path of the
target collection.
[out] outParam1   - A KeyValPair_MS_T containing the results.
[out] outParam2   - an INT_MS_T containing the status.

Description:
Returns the number of objects in a collection by data type.

Note:
This micro-service returns the number of objects for each known data type in a collection, recursively. The results are written to a KeyValPair_MS_T, with the keyword for each data type plus "unknown" for objects of unknown data type, and the number of data objects of that type. The format is:

File-type : Number-of-files

Example Usage:

myTestRule {  
  # Input parameter is:  
  # Path of collection  
  # Output parameters are:  
  # Buffer for results in form key-value pairs  
  # Status  
  msiGetCollectionContentsReport(*Path,*KVPairs,*Status);  
  writeKeyValPairs("stdout",*KVPairs, " : ");  
}  
INPUT *Path="/tempZone/home/rods"  
OUTPUT ruleExecOut
4.195 Module :: ERA :: msiGetCollectionPSmeta

```c
msiGetCollectionPSmeta ( msParam_t * inpParam,
                          msParam_t * outParam    )
```

**Parameters:**

- [in] `inpParam` - A CollInp_MS_T or a STR_MS_T with the iRODS path of the target collection.
- [out] `outParam` - A BUF_LEN_MS_T containing the results.

**Description:**

Retrieves metadata AVU triplets for a collection.

**Note:**

The metadata are returned in a pipe-separated format.

```
Collection-name/Attribute-name/Attribute-value
```

**Example Usage:**

```c
def myTestRule {
    # Input parameter is:
    #  Path of collection
    # Output parameter is:
    #  Buffer holding the result
    msiGetCollectionPSmeta(*Path,*Buf);
    writeLine("stdout","Testing for no metadata on collection");
    writeBytesBuf("stdout",*Buf);
}
```

INPUT *Path="/tempZone/home/rods/sub2"
OUTPUT ruleExecOut
Module :: ERA :: msiGetCollectionSize

**msiGetCollectionSize**

```
(msParam_t * collPath,
    msParam_t * outKVPairs,
    msParam_t * status)
```

**Parameters:**

- **[in]** `collPath` - A CollInp_MS_T or a STR_MS_T with the iRODS path of the target collection.
- **[out]** `outKVPairs` - A KeyValPair_MS_T containing the results.
- **[out]** `status` - an INT_MS_T containing the status.

**Description:**

Returns the object count and total disk usage of a collection.

**Note:**

This micro-service returns the object count and total disk usage for all objects in a collection, recursively. The results are written to a KeyValPair_MS_T whose keyword strings are "Size" and "Object Count".

**Example Usage:**

```c
myTestRule {
    # Input parameter is:
    # Path of the collection
    # Output parameters are:
    # Key-value pairs containing the results
    # Status
    msiGetCollectionSize(*Path,*Buf,*Status);
    msiPrintKeyValPair("stdout",*Buf);
}
```

INPUT *Path="/tempZone/home/rods/ruletest"
OUTPUT ruleExecOut
4.197 Module :: ERA :: msiGetObjectPath

msiGetObjectPath ( msParam_t * object,
                  msParam_t * path,
                  msParam_t * status )

Parameters:
[in] object - A DataObjInp_MS_T, the iRODS object.
[out] path - a STR_MS_T with the object's path.
[out] status - an INT_MS_T containing the operation status.

Description:
Returns the path of an iRODS data object. For use in workflows.

Note:
Returns a STR_MS_T with the object path from a DataObjInp_MS_T structure.

Example Usage:

myTestRule {
    # Input parameter is:
    # Data object structure
    # Output parameters are:
    # Data name converted to a string
    # Status
    *Work="{  
        msiDataObjChksum(*File,"ChksumAll",*chk);
        # convert from object structure to a string for printing
        msiGetObjectPath(*File,*obj,*status);
        writeLine("stdout",'Checksum for *obj is *chk');
    }";
    msiCollectionSpider(*Coll,*File,*Work,*Status);
    writeLine("stdout","Operations completed");
} 

INPUT *Coll="/tempZone/home/rods/sub1"
OUTPUT ruleExecOut
4.198 Module :: ERA :: msiGetDataObjACL

\texttt{msiGetDataObjACL} \ ( \texttt{msParam_t *} \ inpParam, \\
\texttt{msParam_t *} \ outParam ) \\

\textbf{Parameters:}
[-] \texttt{inpParam} - A DataObjInp_MS_T or a STR_MS_T with the iRODS path of the target object.
[-] \texttt{outParam} - A BUF_LEN_MS_T containing the results.

\textbf{Description:}
Gets ACL (Access Control List) for a data object.

\textbf{Note:}
The results are returned in a pipe-separated format:

\texttt{File-name|User-name|Access-control}

The types of access controls are:

<table>
<thead>
<tr>
<th>Access Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>delete metadata</td>
</tr>
<tr>
<td>execute</td>
<td>administer object</td>
</tr>
<tr>
<td>read annotation</td>
<td>create object</td>
</tr>
<tr>
<td>read system metadata</td>
<td>modify object</td>
</tr>
<tr>
<td>read metadata</td>
<td>delete object</td>
</tr>
<tr>
<td>read object</td>
<td>create token</td>
</tr>
<tr>
<td>write annotation</td>
<td>delete token</td>
</tr>
<tr>
<td>create metadata</td>
<td>curate</td>
</tr>
<tr>
<td>modify metadata</td>
<td>own</td>
</tr>
</tbody>
</table>

\textbf{Example Usage:}

\begin{verbatim}
myTestRule { 
  // Input parameter is: 
  // Path of the file 
  // Output parameter is: 
  // Buffer with ACLs
  msiGetDataObjACL(*Path,*Buf);
  writeBytesBuf("stdout",*Buf);
}
INPUT *Path="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
\end{verbatim}
4.199 Module :: ERA :: msiGetDataObjAIP

msiGetDataObjAIP ( msParam_t * inpParam,
                msParam_t * ouputParam )

Parameters:
[in] inpParam - A DataObjInp_MS_T or a STR_MS_T with the iRODS path of
the target object.
[out] outParam - A BUF_LEN_MS_T containing the results.

Description:
This micro-service gets the Archival Information Package of a data object by extracting system level
metadata and descriptive metadata, and packaging them into an XML file.

Note:
The results are returned as an XML file. An Archival Information Package can then be constructed by
packing the XML file and data object into a tar file.

Example Usage:

myTestRule {
  # Input parameter is:
  # Path of file
  # Output parameter is:
  # Buffer listing object metadata in XML format (archival information package)
  msiGetDataObjAIP(*Path,*Buf);
  writeBytesBuf("stdout",*Buf);
}
INPUT *Path="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
4.200 Module :: ERA :: msiGetDataObjAVUs

msiGetDataObjAVUs ( msParam_t * inpParam, msParam_t * outParam )

Parameters:
[in] inpParam - A DataObjInp_MS_T or a STR_MS_T with the iRODS path of the target object.
[out] outParam - A BUF_LEN_MS_T containing the results.

Description:
This micro-service retrieves the metadata AVU triplets for a data object.

Note:
An XML file is created, in the form:

<metadata>
  <AVU>
    <attribute>state</attribute>
    <value>1</value>
    <units></units>
  </AVU>
</metadata>

Each new attribute is encapsulated in <AVU> </AVU> tags.

Example Usage:

myTestRule {
  # Input parameter is:
  # Path of file
  # Output parameter is:
  # Buffer listing AVUs for the file in XML format
  msiGetDataObjAVUs(*Path,*Buf);
  writeBytesBuf("stdout",*Buf);
}

INPUT *Path="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
4.201 Module :: ERA :: msiGetDataObjPSmeta

`msiGetDataObjPSmeta ( msParam_t * inpParam,
                     msParam_t * outParam )`

**Parameters:**

[in] `inpParam` - A DataObjInp_MS_T or a STR_MS_T with the iRODS path of the target object.

[out] `outParam` - A BUF_LEN_MS_T containing the results.

**Description:**

This micro-service retrieves the metadata AVU triplets for a data object

**Note:**

Similar to `msiGetDataObjAVUs` except that the results are returned in a pipe separated format. The output format is:

```
File-path-name|Attribute-name|Attribute-Value|Attribute-Unit
```

**Example Usage:**

```c
myTestRule {
  # Input parameter is:
  # Path of the file
  # Output parameter is:
  # Buffer listing the AVU triplets
  msiGetDataObjPSmeta(*Path,*Buf);
  writeBytesBuf("stdout",*Buf);
}
```

INPUT *Path="/tempZone/home/rods/sub1/foo2"
OUTPUT ruleExecOut
4.202 Module :: ERA :: msiGetUserACL

```c
msiGetUserACL(msParam_t * inpParam1,
  msParam_t * inpParam2,
  msParam_t * outParam )
```

**Parameters:**
- **[in]** `inpParam1` - A STR_MS_T containing the username
- **[in]** `inpParam2` - A BUF_LEN_MS_T containing the ACL results in pipe delimited form
- **[out]** `outParam` - an INT_MS_T containing the status

**Description:**
This micro-service returns a list of all of the files that have ACL permissions for a given iRODS user.

**Note:**
This micro-service also lists files in the trash directory. The output format is:
File-path-name|User-name|Access-control

The types of access controls are:
- `null` delete metadata
- `execute` administer object
- `read annotation` create object
- `read system metadata` modify object
- `read metadata` delete object
- `read object` create token
- `write annotation` delete token
- `create metadata` curate
- `modify metadata` own

**Example Usage:**
```
myTestRule {
  # Input parameter is:
  # User name
  # Output parameters are:
  # Buffer holding ACLs
  # Status
  msiGetUserACL(*User,*Buf,*Status);
  writeBytesBuf("stdout",*Buf);
}
```

INPUT *User="rods"
OUTPUT ruleExecOut
4.203  Module :: ERA :: msiGetUserInfo

msiGetUserInfo ( msParam_t * inpParam1, 
                 msParam_t * inpParam2, 
                 msParam_t * outParam )

Parameters:
[in]  inpParam1   - a msParam of type STR_MS_T with the user name
[in]  inpParam2   - a msParam of type BUF_LEN_MS_T to list user information
[out] outParam    - a msParam of operation status INT_MS_T

Description:
This micro-service returns a given user account's information in pipe-separated format.

Note:
The information includes:
   Username|User-ID|User-role|Zone|info|comment|Create time|Modify time

Example Usage:

myTestRule {  
  # Input parameter is:
  # User
  # Output parameters are:
  # Buffer listing the user account information
  # Status
  msiGetUserInfo(*User,*Buf,*Status);
  writeBytesBuf("stdout",*Buf);
}
INPUT *User="rods"
OUTPUT ruleExecOut
### msiGuessDataType

#### Parameters:
- **inpParam1**: A STR_MS_T containing the object's iRODS path
- **inpParam2**: A STR_MS_T to be filled with the object's data type
- **outParam**: an INT_MS_T containing the status

#### Description:
Guesses the data type of an object based on its file extension.

#### Note:
The path name is not checked to verify the file actually exists. The data types are:

<table>
<thead>
<tr>
<th>AIX DLL</th>
<th>DICOM image</th>
<th>Mac Executable</th>
<th>SGI DLL</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX Executable</td>
<td>directory shadow object</td>
<td>Mac OSX Executable</td>
<td>SGI Executable</td>
</tr>
<tr>
<td>ascii compressed Huffman</td>
<td>DLL</td>
<td>Movie</td>
<td>SGML File</td>
</tr>
<tr>
<td>ascii compressed Lempel-Ziv</td>
<td>Document</td>
<td>MP3 - MPEG Audio</td>
<td>shadow object</td>
</tr>
<tr>
<td>ascii text</td>
<td>DVI format</td>
<td>MPEG</td>
<td>Slide</td>
</tr>
<tr>
<td>audio streams</td>
<td>ebcDIC compressed Huffman</td>
<td>MPEG 3 Movie</td>
<td>Solaris DLL</td>
</tr>
<tr>
<td>AVI</td>
<td>ebcDIC compressed Lempel-Ziv</td>
<td>MPEG Movie</td>
<td>Solaris Executable</td>
</tr>
<tr>
<td>binary file</td>
<td>ebcDIC text</td>
<td>MSWord Document</td>
<td>Spread Sheet</td>
</tr>
<tr>
<td>BMP - Bit Map</td>
<td>email</td>
<td>NSF Award Abstracts</td>
<td>SQL script</td>
</tr>
<tr>
<td>C code</td>
<td>Excel Spread Sheet</td>
<td>NT DLL</td>
<td>streams</td>
</tr>
<tr>
<td>C include file</td>
<td>Executable</td>
<td>NT Executable</td>
<td>tar bundle</td>
</tr>
<tr>
<td>compressed file</td>
<td>fig image</td>
<td>object code</td>
<td>tar file</td>
</tr>
<tr>
<td>compressed mmCIF file</td>
<td>FITS image</td>
<td>orb data</td>
<td>tcl script</td>
</tr>
<tr>
<td>compressed PDB file</td>
<td>fortran code</td>
<td>pbm image</td>
<td>text</td>
</tr>
<tr>
<td>compressed tar file</td>
<td>generic</td>
<td>PDF Document</td>
<td>tiff image</td>
</tr>
<tr>
<td>Cray DLL</td>
<td>gif image</td>
<td>perl script</td>
<td>Troff format</td>
</tr>
<tr>
<td>Cray Executable</td>
<td>html</td>
<td>PNG-Portable Network Graphics</td>
<td>URL</td>
</tr>
<tr>
<td>CSS-Cascading Style Sheet</td>
<td>image</td>
<td>Postscript format</td>
<td>uuencoded tiff</td>
</tr>
<tr>
<td>data file</td>
<td>java code</td>
<td>Power Point Slide</td>
<td>video streams</td>
</tr>
<tr>
<td>database</td>
<td>jpeg image</td>
<td>print-format</td>
<td>Wave Audio</td>
</tr>
<tr>
<td>database object</td>
<td>LaTeX format</td>
<td>program code</td>
<td>WMV-Windows Media Video</td>
</tr>
<tr>
<td>database shadow object</td>
<td>library code</td>
<td>Quicktime Movie</td>
<td>Word format</td>
</tr>
<tr>
<td>datascope data</td>
<td>link code</td>
<td>realAudio</td>
<td>xml</td>
</tr>
<tr>
<td>DICOM header</td>
<td>Mac DLL</td>
<td>realVideo</td>
<td>XML Schema</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example Usage:

myTestRule {
# Input parameter is:
# Path of object
# Output parameters are:
# Data type
# Status
# Output from running the example is:
# File /tempZone/home/rods/DBOtest/lt.pg has data type generic
  msiGuessDataType(*Path,*Type,*Status);
  writeLine("stdout","File *Path has data type *Type");
}
INPUT *Path="/tempZone/home/rods/DBOtest/notthere.mp3"
OUTPUT ruleExecOut
4.205 Module :: ERA :: msiIsColl

**msiIsColl**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in] targetPath</td>
<td>An msParam_t of any type whose inOutStruct is a string (the object's path).</td>
</tr>
<tr>
<td>[out] collId</td>
<td>an INT_MS_T containing the collection ID.</td>
</tr>
<tr>
<td>[out] status</td>
<td>an INT_MS_T containing the operation status.</td>
</tr>
</tbody>
</table>

**Description:**
Checks if an iRODS path is a collection. For use in workflows.

**Note:**
This micro-service takes an iRODS path and returns the corresponding collection ID, or zero if the object is not a collection or does not exist. Avoid path names ending with "/" as they can be misparsed by lower level routines (e.g: use /tempZone/home instead of /tempZone/home/).

**Example Usage:**
```
myTestRule {
    # Input parameter is:
    # Path of collection
    # Output parameters are:
    # Collection ID
    # Status
    # Output from running the example is:
    # The collection ID of /tempZone/home/rods is 10008
    msiIsColl(*Path,*CollID,*Status);
    writeString("stdout","The collection ID of *Path is ");
    writePosInt("stdout",*CollID);
    writeLine("stdout","");
}
```

```
INPUT *Path="/tempZone/home/rods"
OUTPUT ruleExecOut
```
Module :: ERA :: msiIsData

**msiIsData** (msParam_t * targetPath, msParam_t * dataId, msParam_t * status)

**Parameters:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[in]</td>
<td>targetPath</td>
<td>An msParam_t of any type whose inOutStruct is a string (the object's path).</td>
</tr>
<tr>
<td>[out]</td>
<td>dataId</td>
<td>an INT_MS_T containing the data object ID.</td>
</tr>
<tr>
<td>[out]</td>
<td>status</td>
<td>an INT_MS_T containing the operation status.</td>
</tr>
</tbody>
</table>

**Description:**
Checks if an iRODS path is a data object (an iRODS file). For use in workflows.

**Example Usage:**

```plaintext
myTestRule {
    # Input parameter is:
    # Path of file
    # Output parameters are:
    # Data object ID
    # Status
    # Output from running the example is:
    # The data ID of file /tempZone/home/rods/sub1/foo1 is 10077
    msiIsData(*Path,*DataID,*Status);
    writeString("stdout","The data ID of file *Path is ");
    writePosInt("stdout",*DataID);
    writeLine("stdout","\n");
}
INPUT *Path="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
```
4.207 Module :: ERA :: msiLoadACLFromDataObj

msiLoadACLFromDataObj ( msParam_t * inpParam, 
msParam_t * outParam )

Parameters:
[in] inpParam - a msParam of type DataObjInp_MS_T or STR_MS_T with the name of the input file
[out] outParam - status as a msParam of type INT_MS_T

Description:
This micro-service parses an iRODS object for access permissions to update/create.

Note:
The format for ACLs is:
Path-name|user-name|ACL

The types of access controls are:

null       delete metadata
execute    administer object
read annotation    create object
read system metadata    modify object
read metadata    delete object
read object    create token
write annotation    delete token
create metadata    curate
modify metadata    own

Example Usage:

myTestRule {
# Input parameter is:
# File containing desired ACLs has format
# Path-name|user-name|ACL
# /tempZone/home/rods/rodsaccount|guest|read
# Output parameter is:
# Status
  msiLoadACLFromDataObj(*Path,*Status);
  writeLine("stdout","Add ACLs to files from an input file");
}  
INPUT *Path="/tempZone/home/rods/testcoll/rodsaccountACL"
OUTPUT ruleExecOut
4.208 Module :: ERA :: msiLoadMetadataFromDataObj

msiLoadMetadataFromDataObj (msParam_t * inpParam, msParam_t * outParam)

Parameters:
[in] inpParam - A DataObjInp_MS_T or a STR_MS_T with the iRODS path of the metadata file.
[out] outParam - An INT_MS_T containing the status.

Description:
Parses an iRODS object for new metadata AVUs to associate with a file.

Note:
This micro-service parses an iRODS object (file) for new metadata AVUs and adds them to the iCAT. The metadata file is read as a text file and must contain one AVU per line. Collection paths must start with C-. Units in AVUs are optional. Lines starting with # will not be parsed.

Format of the AVU file is:

C-collection-name |Attribute |Value |Units
Path-name-for-file |Attribute |Value |Units

Example Usage:

myTestRule {
# 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
# /tempZone/home/rods/sub1/foo1 |Test |34
# Output parameter is:
# Status
msiLoadMetadataFromDataObj(*Path,*Status);
msiGetDataObjAVUs(*Filepath,*Buf);
writeBytesBuf("stdout",*Buf);
}
INPUT *Path="/tempZone/home/rods/testcoll/rodsaccountavus",
*Filepath="/tempZone/home/rods/sub1/foo1"
OUTPUT ruleExecOut
**4.209 Module :: ERA :: msiLoadUserModsFromDataObj**

```c
msiLoadUserModsFromDataObj (msParam_t * inpParam, 
                             msParam_t * outParam)
```

**Parameters:**
- [in] `inpParam` - a `msParam` of type `DataObjInp_MS_T` or `STR_MS_T` with the name of the account file
- [out] `outParam` - a `msParam` of operation status `INT_MS_T`

**Description:**
This micro-service modifies existing user accounts according to information in an iRODS object.

**Note:**
The format of the account file is:
```plaintext
user-name|field|new-value
```
where valid fields include:
- type
- zone
- comment
- info
- password

**Example Usage:**
```c
myTestRule {
    # Input parameter is:
    # Path of file containing information
    # Output parameter is:
    # Format of the file is
    # user-name|field|new-value
    # guest|password|guest1
    # Status
    msiLoadUserModsFromDataObj(*Path,*Status);
    writeLine("stdout","Change password on a user account");
}
```

INPUT *Path="/tempZone/home/rods/testcoll/rodsaccountmod"
OUTPUT ruleExecOut
4.210  Module :: ERA :: msiMergeDataCopies

**msiMergeDataCopies**  
(  
msParam_t* objPath,  
msParam_t* currentColl,  
msParam_t* masterColl,  
msParam_t* status  )  

**Parameters:**  
[in]  
objPath - A DataObjInp_MS_T or a STR_MS_T with the target object's path.  
[in]  
currentColl - A CollInp_MS_T or a STR_MS_T with the current collection's path.  
[in]  
masterColl - A CollInp_MS_T or a STR_MS_T with the master collection's path.  
[out]  
status - An INT_MS_T containing the operation status.  

**Description:**  
Custom micro-service for NARA consolidation rule. This consolidates multiple collections, to create an authoritative collections that contains all of the files.  

**Note:**  
Takes an object, a current home collection and a master collection:  
1) If the object is an orphan (no corresponding object in the master collection) it is moved to the master collection.  
2) If there is a corresponding object in the master collection, their checksums are compared.  
2a) If the checksums match, the object is registered as a replica of the master object. Its physical file is rearranged accordingly to ensure consistency between logical and physical paths.  
2b) If the checksums do not match, the object is flagged for manual check: It gets a new metadata attribute whose name is "CHECKSUM_MISMATCH" and whose value is the checksum of the master object.  

**Example Usage:**  

```
myTestRule {  
# Input parameters are:  
# File  
# Current collection path  
# Master collection path  
# Note that the directory structures within the current collection are mirrored into the master  
# Merges one file at a time, if file already exists creates a replica  
# The microservice uses two commands in iRODS/server/bin/cmd:  
# ln - softlink  
# mkdir - make directory  
# Output parameter is:  
# Status  
# Output from running example is:  
# List of files in the master directory before the merge  
# List of files in the master directory after the merge  
msiSplitPath(*Path,*Coll,*Fileorig);  
writeLine("stdout","Collection *Master has initial files");  
msiExecStrCondQuery("SELECT DATA_NAME,DATA_ID where COLL_NAME = '*Master'",*QOut);  
foreach (*QOut) {  
  msiGetValByKey(*QOut,"DATA_NAME",*File);  
  writeLine("stdout",*File);  
}  
msiMergeDataCopies(*Path,*Coll,*Master,*Status);  
writeLine("stdout","Collection *Master has files after merge");  
msiExecStrCondQuery("SELECT DATA_NAME,DATA_ID where COLL_NAME = '*Master'",*QOut);  
```
foreach (*QOut) {
    msiGetValByKey(*QOut,"DATA_NAME",*File);
    writeLine("stdout",*File);
}

INPUT *Path="/tempZone/home/rods/sub1/foo1", *Master="/tempZone/home/rods/test"
OUTPUT ruleExecOut
4.211 Module :: ERA :: msiRecursiveCollCopy

```c
msiRecursiveCollCopy ( msParam_t * inpParam1,
                      msParam_t * inpParam2,
                      msParam_t * outParam)
```

**Parameters:**
- **inpParam1** - A CollInp_MS_T or a STR_MS_T with the iRODS path of the destination collection.
- **inpParam2** - A CollInp_MS_T or a STR_MS_T with the iRODS path of the source collection.
- **outParam** - An INT_MS_T containing the status.

**Description:**
Copies a collection and its contents recursively.

**Note:**
Collection and data object metadata AVUs are also copied over to the new objects and collections.

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  #  Path of the destination collection
  #  Path of the source collection
  # Output parameter is:
  #  Status
  writeLine("stdout","Files in destination collection before copy are:");
  msiExecStrCondQuery("SELECT DATA_NAME where COLL_NAME = "DestColl"",*QOut);
  foreach(*QOut) {
    msiGetValByKey(*QOut,"DATA_NAME",*File);
    writeLine("stdout","*File");
  }
  msiRecursiveCollCopy(*DestColl,*SourceColl,*Status)
  writeLine("stdout","Files in destination collection after copy are:");
  msiExecStrCondQuery("SELECT DATA_NAME where COLL_NAME = "DestColl"",*QOut);
  foreach(*QOut) {
    msiGetValByKey(*QOut,"DATA_NAME",*File);
    writeLine("stdout","*File");
  }
}
```

INPUT *SourceColl="/tempZone/home/rods/sub1", *DestColl="/tempZone/home/rods/sub2"
OUTPUT ruleExecOut
4.212 Module :: ERA :: msiSetDataType

\textbf{msiSetDataType}( \textit{msParam\_t*\ inpParam1},
\textit{msParam\_t*\ inpParam2},
\textit{msParam\_t*\ inpParam3},
\textit{msParam\_t*\ outParam})

\textbf{Parameters:}
\begin{itemize}
  \item \textbf{[in]} \textit{inpParam1} - A STR\_MS\_T containing the object ID
  \item \textbf{[in]} \textit{inpParam2} - A STR\_MS\_T containing the object's iRODS path
  \item \textbf{[in]} \textit{inpParam3} - A STR\_MS\_T containing the data type to be set
  \item \textbf{[out]} \textit{outParam} - an INT\_MS\_T containing the status
\end{itemize}

\textbf{Description:}
Sets the data\_type\_name attribute of a data object.

\textbf{Note:}
The allowed data types are:

\begin{tabular}{|c|c|c|c|}
\hline
AIX DLL & DICOM image & Mac Executable & SGI DLL \\
\hline
AIX Executable & directory shadow object & Mac OSX Executable & SGI Executable \\
\hline
ascii compressed Huffman DLL & Movie & SGML File & \\
\hline
ascii compressed Huffman Document & MP3 - MPEG Audio & shadow object & \\
\hline
ascii text & DVI format & MPEG & Slide \\
\hline
audio streams & ebcdic compressed Huffman MPEG 3 Movie & Solaris DLL & \\
\hline
AVI & ebcdic compressed Lempel-Ziv MPEG Movie & Solaris Executable & \\
\hline
binary file & ebcdic text & MSWord Document & Spread Sheet \\
\hline
BMP - Bit Map & email & NSF Award Abstracts & SQL script \\
\hline
C code & Excel Spread Sheet & NT DLL & streams \\
\hline
C include file & Executable & NT Executable & tar bundle \\
\hline
compressed file & fig image & object code & tar file \\
\hline
compressed mmCIF file & FITS image & orb data & tcl script \\
\hline
compressed PDB file & fortran code & pbm image & text \\
\hline
compressed tar file & generic & PDF Document & tiff image \\
\hline
Cray DLL & gif image & perl script & Troff format \\
\hline
Cray Executable & html & PNG-Portable Network Graphics & URL \\
\hline
CSS-Cascading Style Sheet & image & Postscript format & uuencoded tiff \\
\hline
data file & java code & Power Point Slide & video streams \\
\hline
database & jpeg image & print-format & Wave Audio \\
\hline
database object & LaTeX format & program code & WMV-Windows Media Video \\
\hline
database shadow object & library code & Quicktime Movie & Word format \\
\hline
datascope data & link code & realAudio & xml \\
\hline
DICOM header & Mac DLL & realVideo & XML Schema \\
\hline
\end{tabular}
Example Usage:

myTestRule {
    # Input parameters are:
    # File ID
    # File path
    # Data Type to set
    # Output parameter is:
    # Status

    msiSplitPath(*Path, *Coll, *File);
    msiExecStrCondQuery("SELECT DATA_ID where COLL_NAME = '*Coll' and DATA_NAME = '*File', *QOut);
    foreach(*QOut) {
        msiGetValByKey(*QOut, "DATA_ID", *Objid);
        msiSetDataType(*Objid, *Path, *Datatype, *Status);
    }
    msiExecStrCondQuery("SELECT DATA_TYPE_NAME where COLL_NAME = '*Coll' and DATA_NAME = '*File', *QOut1);
    foreach(*QOut1) {
        msiGetValByKey(*QOut1, "DATA_TYPE_NAME", *Dtype);
        writeLine("stdout", "Data type retrieved is *Dtype");
    }
}

INPUT *Path="/tempZone/home/rods/sub1/foo1", *Datatype="text"
OUTPUT ruleExecOut
4.213  Module :: ERA :: msiStripAVUs

**msiStripAVUs** (  
msParam_t*  target,  
msParam_t*  options,  
msParam_t*  status )

**Parameters:**
- **[in]** target - A STR_MS_T with a data/collection path or user/resource name
- **[in]** options - Optional - a STR_MS_T that contains one of more options in the format
  keyWd1=value1++++keyWd2=value2++++keyWd3=value3... The type of target will be specified here (i.e. data, collection, user, resource, etc...).
  Valid keywords will be added as more types are supported.
- **[out]** status - An INT_MS_T containing the operation status.

**Description:**
Strips an object of its metadata.

**Note:**
Only data objects are supported for now.

**Example Usage:**

```c
myTestRule {
    # Input parameter are:
    # Path to data file
    # Optional flag - not used
    # Output parameter is:
    # Status

    # Set an AVU
    msiFlagDataObjwithAVU(*Path,"State",*Status);

    # Retreive the AVUs
    msiGetDataObjPSmeta(*Path,*Buf);
    writeLine("stdout","Metadata on *Path is:");
    writeBytesBuf("stdout",*Buf);
    writeLine("stdout","" );

    # Delete the AVUs
    msiStripAVUs(*Path,"",*Status);

    # Verify the AVUs have been deleted
    msiGetDataObjPSmeta(*Path,*Buf);
    writeLine("stdout","Metadata has been removed on *Path");
    writeBytesBuf("stdout",*Buf);
    writeLine("stdout","" );
}
```

**INPUT** *Path="/tempZone/home/rods/sub1/foo2"
**OUTPUT** ruleExecOut
4.214 Module :: ERA :: msiStructFileBundle

`msiStructFileBundle` (
    msParam_t * collection,
    msParam_t * bundleObj,
    msParam_t * resource,
    msParam_t * status)

**Parameters:**

- [in] `collection` - A CollInp_MS_T or a STR_MS_T with the iRODS path of the collection to bundle.
- [in] `bundleObj` - a DataObjInp_MS_T or a STR_MS_T with the bundle object's path.
- [in] `resource` - Optional - a STR_MS_T which specifies the target resource.
- [out] `status` - an INT_MS_T containing the operation status.

**Description:**
Bundles a collection for export

**Note:**
This micro-service creates a bundle for export from an iRODS collection on a target resource. Files in the collection are first replicated onto the target resource. If no resource is given the default resource will be used.

**Example Usage:**

```plaintext
myTestRule {
    # Input parameters are:
    # Path of collection to bundle
    # Path of bundle that will be created
    # Optional target resource
    # Output parameter is:
    # Status
    # Output from running the example is:
    # Bundle file created as a tar file
    # Bundle file created as a tar file
    msiStructFileBundle(*Path,*Bundle,*Resc,*Status);
    msiGuessDataType(*Bundle,*Type,*Status);
    writeLine("stdout","Bundle file created as a *Type");
}
```

INPUT *Path="/tempZone/home/rods/sub1", *Bundle="/tempZone/home/rods/test/sub1.tar",
*Resc="testResc"
OUTPUT ruleExecOut
4.215 Module :: Guinot :: msiGetFormattedSystemTime

```c
msiGetFormattedSystemTime (msParam_t * outParam,
                           msParam_t * inpParam,
                           msParam_t * inpFormatParam)
```

**Parameters:**
- **[out]** `outParam` - a STR_MS_T containing the time
- **[in]** `inpParam` - Optional - a STR_MS_T containing the desired output format (human)
- **[in]** `inpFormatParam` - Optional - a STR_MS_T containing printf formatting (if `inpParam` was "human")

**Description:**
This micro-service returns the local system time.

**Note:**
This micro-service is deprecated. Use `msiGetIcatTime` instead. Default output format is system time in seconds since unix epoch, use "human" as input parameter for human readable format.

**Example Usage:**

```plaintext
myTestRule {
  # Input parameters are:
  # Optional format flag - human
  # Optional snprintf formatting for human format, using six inputs (year,month,day,hour,minute,second)
  # Output parameter is:
  # Local system time
  msiGetFormattedSystemTime(*Out,"null","null");
  writeLine("stdout",*Out);
}
```

INPUT null
OUTPUT ruleExecOut

```plaintext
rulemsiGetFormattedSystemTime-human.r
myTestRule {
  # Input parameters are:
  # Optional format flag - human
  # Optional snprintf formatting for human format, using six inputs (year,month,day,hour,minute,second)
  # - the example below will print an ISO 8601 extended format datetimestamp
  # Output parameter is:
  # Local system time
  msiGetFormattedSystemTime(*Out,"human","%d-%02d-%02dT%02d:%02d:%02d");
  writeLine("stdout",*Out);
}
```

INPUT null
OUTPUT ruleExecOut
4.216  Module :: msoDrivers :: msiobjget_dbo

**msiobjget_dbo** (  
msParam_t * inRequestPath,  
msParam_t * inFileMode,  
msParam_t * inFileFlags,  
msParam_t * inCacheFilename )

**Parameters:**

- **[in]** inRequestPath - a STR_MS_T containing the request sent to the external resource.
- **[in]** inFileMode - a STR_MS_T containing the mode for the cache file creation.
- **[in]** inFileFlags - a STR_MS_T containing access flags for cache file creation.
- **[in]** inCacheFilename - a STR_MS_T containing the full path for the local cache file name.

**Description:**

This micro-service gets a database object from a database resource that has been registered into the iRODS data grid. The object is stored in the cache filename.

**Note:**

The inFileMode argument specifies the permissions to use when creating the cache file:

- r or rb: Open file for reading.
- w or wb: Truncate to zero length or create file for writing.
- a or ab: Append; open or create file for writing at end-of-file.
- r+ or rb+ or r+b: Open file for update (reading and writing).
- w+ or wb+ or w+b: Truncate to zero length or create file for update.
- a+ or ab+ or a+b: Append; open or create file for update, writing at end-of-file.

The inFileFlags argument specifies the access mode for the cache file:

- O_RDONLY,
- O_WRONLY,
- O_RDWR
- O_TRUNC.

These can be combined by concatenation, e.g. O_WRONLYO_TRUNC.

The inRequestPath starts with "dbo:". The string after that has two parts separated by a colon, ":". The first part is the DB Resource Name. The second part is the DBO file name in iRODS. An example is:  

dbo:dbr2:/tempZone/home/rods/dbotest/lt.pg

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  # inRequestPath - the string sent to the remote database object
  # inFileMode - the cache file creation mode
  # inFileFlags - the access modes for the cache file
  # inCacheFilename - the full path of the cache file
  # No output parameters
  # Output is the name of the file that was created
  msiobjget_dbo(*Request, *Mode, *Flags, *Path);
}
INPUT *Request = "dbo:dbr2:/tempZone/home/rods/dbotest/lt.pg", *Mode = "w", *Flags = "O_RDWR",
  *Path = "/home/reagan/Vault/home/rods/sub1/rodsfile"
OUTPUT ruleExecOut
```
4.217  Module :: msoDrivers :: msiobjget_http

msiobjget_http ( msParam_t * inRequestPath,
                 msParam_t * inFileMode,
                 msParam_t * inFileFlags,
                 msParam_t * inCacheFilename )

Parameters:

[in] inRequestPath   - a STR_MS_T containing the request sent to the external resource.
[in] inFileMode      - a STR_MS_T containing the mode for the cache file creation.
[in] inFileFlags     - a STR_MS_T containing access flags for cache file creation.
[in] inCacheFilename - a STR_MS_T containing the full path for the local cache filename.

Description:

This micro-service gets a web object from a URL using micro-service drivers. The object is stored in the cache file name on the local resource. The web object can be an http, https or ftp object.

Note:

The inFileMode argument specifies the permissions to use when creating the cache file:

- r or rb: Open file for reading.
- w or wb: Truncate to zero length or create file for writing.
- a or ab: Append; open or create file for writing at end-of-file.
- r+ or rb+ or r+b: Open file for update (reading and writing).
- w+ or wb+ or w+b: Truncate to zero length or create file for update.
- a+ or ab+ or a+b: Append; open or create file for update, writing at end-of-file.

The inFileFlags argument specifies the access mode for the cache file:

- O_RDONLY,
- O_WRONLY,
- O_RDWR
- O_TRUNC.

These can be combined by concatenation, e.g. O_WRONLYO_TRUNC

The inRequestPath starts with "http:", "https:", or "ftp:". The string after that is the iRODS logical path name. Examples are:
- http://farm3.static.flickr.com/2254/5827459234_2fd1c5536_z.jpg
- ftp://ftp.sdsc.edu/pub/outgoing/sekar/PPP.txt
- https://www.irods.org/index.php

Example Usage:

myTestRule {
  # Input parameters are:
  # inRequestPath - the string sent to the remote URL
  # inFileMode - the cache file creation mode
  # inFileFlags - the access modes for the cache file
  # inCacheFilename - the full path of the cache file on the local system
  # No output parameters
  # Output is the creation of a file in the vault
  # Wrote local file /home/reagan/Vaulttest/webfile from request http://irods.org.pubs/iRODS_FACT_Sheet-0907c.pdf
  msiobjget_http(*Request, *Mode, *Flags, *Path);
  writeLine("stdout","Wrote local file *Path from request *Request");
}
4.218 Module :: msoDrivers :: msiobjget_irods

msiobjget_irods (msParam_t * inRequestPath,
msParam_t * inFileMode,
msParam_t * inFileFlags,
msParam_t * inCacheFilename)

Parameters:
[in] inRequestPath - a STR_MS_T containing the request sent to the external resource.
[in] inFileMode - a STR_MS_T containing the mode for the cache file creation.
[in] inFileFlags - a STR_MS_T containing access flags for cache file creation.
[in] inCacheFilename - a STR_MS_T containing the full path for the local cache filename.

Description:
This micro-service gets an iRODS object from a remote iRODS data grid using micro-service drivers. The object is stored in the local cache filename.

Note:
The inFileMode argument specifies the permissions to use when creating the cache file:
- r or rb: Open file for reading.
- w or wb: Truncate to zero length or create file for writing.
- a or ab: Append; open or create file for writing at end-of-file.
- r+ or rb+ or r+b: Open file for update (reading and writing).
- w+ or wb+ or w+b: Truncate to zero length or create file for update.
- a+ or ab+ or a+b: Append; open or create file for update, writing at end-of-file.

The inFileFlags argument specifies the access mode for the cache file:
- O_RDONLY,
- O_WRONLY,
- O_RDWR
- O_TRUNC.
These can be combined by concatenation, e.g. O_WRONLY O_TRUNC.

The inRequestPath starts with "irods:" The string after that has three parts separated by a colon ":".
- The first part is the iRODS host name.
- The second part is the iRODS port number.
- The third part consists of two sections,
  - a user name,
  - an iRODS logical pathname.

The user name can have a zone name delineated by @.

For LOCALZONE anonymous access:
  irods:srbbrick14.sdsc.edu:2247:anonymous/tempZone/home/rods/mytest/irm.c

For NON-FEDERATED REMOTE ZONE access:
  irods:iren.renci.org:1247:anonymous@renci/renci/home/rods/README.txt

Example Usage:

```
myTestRule {
  # Input parameters are:
  # inRequestPath - the string sent to the remote iRODS data grid
  # inFileMode - the cache file creation mode
  # inFileFlags - the access modes for the cache file
  # inCacheFilename - the full path of the cache file
  # No output parameters
  # Output is the creation of a file on the local vault
  msiobjget_irods(*Request, *Mode, *Flags, *Path);
}
```

```c
INPUT *Request ="irods:iren.renci.org:1247:anonymous@renci/renci/home/rods/README.txt", *Mode =
```
"w", *Flags = "O_RDWR", *Path = "/home/reagan/Vaulttest/home/rods/sub1/rodsfile"
OUTPUT ruleExecOut
Module :: msoDrivers :: msiobjget_slink

```c
msiobjget_slink (msParam_t * inRequestPath,
                msParam_t * inFileMode,
                msParam_t * inFileFlags,
                msParam_t * inCacheFilename)
```

**Parameters:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inRequestPath</code></td>
<td>a STR_MS_T containing the request sent to the external resource.</td>
</tr>
<tr>
<td><code>inFileMode</code></td>
<td>a STR_MS_T containing the mode for the cache file creation.</td>
</tr>
<tr>
<td><code>inFileFlags</code></td>
<td>a STR_MS_T containing access flags for cache file creation.</td>
</tr>
<tr>
<td><code>inCacheFilename</code></td>
<td>a STR_MS_T containing the full path for the local cache filename.</td>
</tr>
</tbody>
</table>

**Description:**

This micro-service gets an iRODS object from a soft link to another iRODS data grid using micro-service drivers. The object is stored in the cache filename.

**Note:**

The `inFileMode` argument specifies the permissions to use when creating the cache file:

- `r` or `rb`: Open file for reading.
- `w` or `wb`: Truncate to zero length or create file for writing.
- `a` or `ab`: Append; open or create file for writing at end-of-file.
- `r+` or `rb+` or `r+b`: Open file for update (reading and writing).
- `w+` or `wb+` or `w+b`: Truncate to zero length or create file for update.
- `a+` or `ab+` or `a+b`: Append; open or create file for update, writing at end-of-file.

The `inFileFlags` argument specifies the access mode for the cache file:

- `O_RDONLY`,
- `O_WRONLY`,
- `O_RDWR`,
- `O_TRUNC`.

These can be combined by concatenation, e.g. `O_RDONLYO_TRUNC`.

The `inRequestPath` starts with "slink:". The string after that is the iRODS logical path name. An example is:

```
slink:/tempZone/home/rods/mytest/iinit.c
```

**Example Usage:**

```c
myTestRule {}
# Input parameters are:
# inRequestPath - the string sent to the remote iRODS data grid
# inFileMode - the cache file creation mode
# inFileFlags - the access modes for the cache file
# inCacheFilename - the full path of the cache file
# No output parameters
# Output is the creation of a file on the local cache
  msiobjget_slink(*Request, *Mode, *Flags, *Path);
}
```

```
INPUT *Request ="slink:/renci/home/rods/README.txt", *Mode = "w", *Flags = "O_RDWR", *Path = "/home/reagan/Vaulttest/home/rods/sub1/rodsfile"
OUTPUT ruleExecOut
```
4.220 Module :: msoDrivers :: msiobjget_srb

**msiobjget_srb** ( 
  msParam_t * inRequestPath,
  msParam_t * inFileMode,
  msParam_t * inFileFlags,
  msParam_t * inCacheFilename )

**Parameters:**

[in] inRequestPath - a STR_MS_T containing the request sent to the external resource.
[in] inFileMode - a STR_MS_T containing the mode for the cache file creation.
[in] inFileFlags - a STR_MS_T containing access flags for cache file creation.
[in] inCacheFilename - a STR_MS_T containing the full path for the local cache filename.

**Description:**

This micro-service gets a file from a Storage Resource Broker data grid. The object is stored in the cache filename.

**Note:**

The inFileMode argument specifies the permissions to use when creating the cache file:

- r or rb: Open file for reading.
- w or wb: Truncate to zero length or create file for writing.
- a or ab: Append; open or create file for writing at end-of-file.
- r+ or rb+ or r+b: Open file for update (reading and writing).
- w+ or wb+ or w+b: Truncate to zero length or create file for update.
- a+ or ab+ or a+b: Append; open or create file for update, writing at end-of-file.

The inFileFlags argument specifies the access mode for the cache file:

- O_RDONLY
- O_WRONLY
- O_RDWR
- O_TRUNC.

These can be combined by concatenation, e.g. O_WRONLYO_TRUNC

The inRequestPath starts with "srb:" The string after that has three parts separated by a colon, ":".
- The first part is the SRB host name.
- The second part is the SRB port number.
- The third part consists of two sections,
  - A user name,
  - An SRB logical pathname.

The user name can have a zone name delineated by @.

An example is:

```
srb:srbbrick11.sdsc.edu:7676:testuser@sdsc:TESTUSER/UCHRI/home/srbAdmin.uchri/testdir/testFile
```

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  # inRequestPath - the string sent to the remote Storage Resource Broker data grid
  # inFileMode - the cache file creation mode
  # inFileFlags - the access modes for the cache file
  # inCacheFilename - the full path of the cache file
  # No output parameters
```
# Output is the creation of a file on the local vault

```c
msiobjget_srb(*Request, *Mode, *Flags, *Path);
```

```c
}
```

```c
INPUT *Path = "/home/reagan/Vaulttest/home/rods/sub1/rodsfile",
*Request="srb:srbbrick11.sdsc.edu:7676:testuser@sdsc:TESTUSER/UCHRI/home/srbAdmin.uchri/testdir/
testFile", *Mode = "w", *Flags = "O_RDWR"
OUTPUT ruleExecOut
```
4.221 Module :: msoDrivers :: msiobjget_test

msiobjget_test ( msParam_t * inRequestPath,
                msParam_t * inFileMode,
                msParam_t * inFileFlags,
                msParam_t * inCacheFilename )

Parameters:
[in]  inRequestPath - a STR_MS_T containing the request for testing the micro-service object framework
[in]  inFileMode - a STR_MS_T containing the mode for the cache file creation.
[in]  inFileFlags - a STR_MS_T containing access flags for cache file creation.
[in]  inCacheFilename - a STR_MS_T containing the full path for the cache file.

Description:
This micro-service tests the micro-service object framework. It creates a file on the local vault to represent the creation of a cache file to hold the result of accessing a remote resource through a soft link.

Note:
The inFileMode argument specifies the permissions to use when creating the cache file:
   r or rb Open file for reading.
   w or wb Truncate to zero length or create file for writing.
   a or ab Append; open or create file for writing at end-of-file.
   r+ or rb+ or r+b Open file for update (reading and writing).
   w+ or wb+ or w+b Truncate to zero length or create file for update.
   a+ or ab+ or a+b Append; open or create file for update, writing at end-of-file.

The inFileFlags argument specifies the access mode for the cache file:
   O_RDONLY,
   O_WRONLY,
   O_RDWR
   O_TRUNC.
These can be combined by concatenation, e.g. O_WRONLYO_TRUNC

The inRequestPath starts with "test:". An example is:
test:Test string

Example Usage:

myTestRule {
   # Input parameters are:
   # inRequestPath - the string defining a read from a file in iRODS
   # inFileMode - the cache file creation mode
   # inFileFlags - the access modes for the cache file
   # inCacheFilename - the full path of the cache file
   # No output parameters
   # Output
   # The specified file contains “PID is pid-number. This is a test”
   # You will need to set the access permission on the file, chmod a+r filename
       msiobjget_test(*Request, *Mode, *Flags, *Path);
}

INPUT *Request ="test:Test string", *Mode = "r+", *Flags = "O_RDWR", *Path = "/home/reagan/Vault/home/rods/rodsfile"
OUTPUT ruleExecOut
4.222 Module :: msoDrivers :: msiobjget_z3950

```
msiobjget_z3950( msParam_t * inRequestPath,
               msParam_t * inFileMode,
               msParam_t * inFileFlags,
               msParam_t * inCacheFilename )
```

**Parameters:**

- **inRequestPath** - a STR_MS_T containing the request sent to the external resource.
- **inFileMode** - a STR_MS_T containing the mode for the cache file creation.
- **inFileFlags** - a STR_MS_T containing access flags for cache file creation.
- **inCacheFilename** - a STR_MS_T containing the full path for the local cache filename.

**Description:**

This micro-service gets data from a Z39.50 source using micro-service drivers. The object is stored in the cache filename.

**Note:**

The inFileMode argument specifies the permissions to use when creating the cache file:

- **r** or **rb** - Open file for reading.
- **w** or **wb** - Truncate to zero length or create file for writing.
- **a** or **ab** - Append; open or create file for writing at end-of-file.
- **r+** or **rb+** or **r+b** - Open file for update (reading and writing).
- **w+** or **wb+** or **w+b** - Truncate to zero length or create file for update.
- **a+** or **ab+** or **a+b** - Append; open or create file for update, writing at end-of-file.

The inFileFlags argument specifies the access mode for the cache file:

- **O_RDONLY**
- **O_WRONLY**
- **O_RDWR**
- **O_TRUNC**

These can be combined by concatenation, e.g. **O_WRONLYO_TRUNC**

The inRequestPath starts with "z3950". The string after that is a Z39.50 compliant request string in either the MARC syntax or an XML syntax. Examples are:

- **MARC syntax**:
  ```
  z3950:z3950.loc.gov:7090/Voyager?query=@attr 1=1003 Marx&recordsyntax=USMARC
  ```
- **XML syntax**:
  ```
  z3950:z3950.loc.gov:7090/Voyager?query=@attr 1=1003 Marx
  ```

**Example Usage:**

```bash
myTestRule {
  # Input parameters are:
  # inRequestPath - the string sent to the remote Z39.50 site
  # inFileMode - the cache file creation mode
  # inFileFlags - the access modes for the cache file
  # inCacheFilename - the full path of the cache file
  # No output parameters
  # Output is the name of the file that was created
  msiobjget_z3950(*Request, *Mode, *Flags, *Path);
}
```

```
INPUT *Request ="z3950:z3950.loc.gov:7090/Voyager?query=@attr 1=1003 Marx"
```
Marx\&recordsyntax=USMARC", *Mode = "w", *Flags = "O_RDWR", *Path = "/home/reagan/Vaulttest/home/rods/sub1/rodsfile"
OUTPUT ruleExecOut
4.223 Module :: msoDrivers :: msiobjput_dbo

msiobjput_dbo (     msParam_t * inMSOPath,
                    msParam_t * inCacheFilename,
                    msParam_t * inFileSize     )

Parameters:
[in] inMSOPath - a STR_MS_T containing the request sent to the external resource.
[in] inCacheFilename - a STR_MS_T containing the full path for the local cache filename to be written out.
[in] inFileSize - a STR_MS_T containing the size of the cache file.

Description:
This micro-service puts information into a database object that has been registered into an iRODS data grid.

Note:
The inMSOPath starts with "dbo:". The string after that has two parts separated by a colon, ":". The first part is the DB Resource Name. The second part is the DBO file name in iRODS. An example is:
   dbo:dbr2:/tempZone/home/rods/dbotest/lt.pg

Example Usage:

myTestRule {
    # Input parameters are:
    # inMSOPath - the string sent to the remote database object
    # inCacheFilename - the full path of the cache file
    # inFileSize - the size of the cache file, found from ls on the vault
    # No output parameters
    msiobjput_dbo(*Request, *Path, *Size);
}

INPUT *Request ="dbo:dbr2:/tempzone/home/rods/dbotest/lt.pg", *Path = "/home/reagan/Vaulttest/home/rods/sub1/rodsfile", *Size = "15"
OUTPUT ruleExecOut
4.224 Module :: msoDrivers :: msiobjput_http

**msiobjput_http** (msParam_t * inMSOPath,
msParam_t * inCacheFilename,
msParam_t * inFileSize)

**Parameters:**
- [in] **inMSOPath** - a STR_MS_T containing the request sent to the external resource.
- [in] **inCacheFilename** - a STR_MS_T containing the full path for the local cache filename to be written out.
- [in] **inFileSize** - a STR_MS_T containing the size of the cache file.

**Description:**
This micro-service puts an http object file using micro-service drivers. The object is written to the remote URL.

**Note:**
The inMSOPath starts with "http:", "https:“, or "ftp:“. The string after that is the iRODS logical path name.
Examples are:
- http://farm3.static.flickr.com/2254/5827459234_2fd1c55364_z.jpg
- ftp://ftp.sdsc.edu/pub/outgoing/sekar/PPP.txt
- https://www.irods.org/index.php

**Example Usage:**
myTestRule {
# Input parameters are:
# inMSOPath - the string sent to the remote URL
# inCacheFilename - the full path of the cache file
# inFileSize - the size of the cache file, found from ls on the vault
# No output parameters
  msiobjput_http(*Request, *Path, *Size);
}

INPUT *Request ="http://farm3.static.flickr.com/2254/5827459234_2fd1c55364_z.jpg", *Path = "/tempZone/home/rods/sub1/rodsfile", *Size = "15"
OUTPUT ruleExecOut
4.225  Module :: msoDrivers :: msiobjput_irods

msiobjput_irods ( msParam_t * inMSOPath,
    msParam_t * inCacheFilename,
    msParam_t * inFileSize )

Parameters:
[in] inMSOPath - a STR_MS_T containing the request sent to the external resource.
[in] inCacheFilename - a STR_MS_T containing the full path for the local cache filename to be written out.
[in] inFileSize - a STR_MS_T containing the size of the cache file.

Description:
This micro-service puts an iRODS object into a remote iRODS data grid using micro-service drivers.
The object is written to the remote resource.

Note:
The inMSOPath starts with "irods:" The string after that has three parts separated by a colon ":".
   The first part is the iRODS host name.
   The second part is the iRODS port number.
   The third part consists of two sections,
      a user name,
      an iRODS logical pathname.
The user name can have a zone name delineated by @.
For LOCALZONE anonymous access:
   irods:srbbrick14.sdsc.edu:2247:anonymous/tempZone/home/rods/mytest/irm.c
For NON-FEDERATED REMOTE ZONE access:
   irods:iren.renci.org:1247:anonymous@renci/renci/home/rods/README.txt

Example Usage:

myTestRule {  
    # Input parameters are:
    # Input parameters are:
    # Input parameters are:
    # inMSOPath - the string sent to the remote iRODS data grid
    # inCacheFilename - the full path of the cache file
    # inFileSize - the size of the cache file
    # No output parameters
    msiobjput_irods(*Request, *Path, *Size);  
}  
INPUT *Request ="irods:iren.renci.org:1247:anonymous@renci/renci/home/rods/README.txt", *Path =
"/home/reagan/Vaulttest/home/rods/sub1/rodsfile", *Size = "15"
OUTPUT ruleExecOut
4.226 Module :: msoDrivers :: msiobjput_slink

```c
msiobjput_slink ( msParam_t * inMSOPath,
                  msParam_t * inCacheFilename,
                  msParam_t * inFileSize )
```

**Parameters:**
- **[in]** `inMSOPath` - a STR_MS_T containing the request sent to the external resource.
- **[in]** `inCacheFilename` - a STR_MS_T containing the full path for the local cache filename to be written out.
- **[in]** `inFileSize` - a STR_MS_T containing the size of the cache file.

**Description:**
This micro-service puts an iRODS object into a remote iRODS data grid through a soft link using micro-service drivers. The object is written to the remote resource.

**Note:**
The `inMSOPath` starts with "slink:" The string after that is an iRODS logical path name. An example is:

```
slink:/tempZone/home/rods/mytest/iinit.c
```

**Example Usage:**

```c
define myTestRule {
    # Input parameters are:
    #  inMSOPath - the string sent to the remote iRODS data grid
    #  inCacheFilename - the full path of the cache file
    #  inFileSize - the size of the cache file
    # No output parameters
    msiobjput_slink(*Request, *Path, *Size);
}
```

INPUT *Request ="slink:/renci/home/rods/README.txt", *Path = "/home/reagan/Vaulttest/home/rods/sub1/rodsfile", *Size = "15"
OUTPUT ruleExecOut
4.227 Module :: msoDrivers :: msiobjput_srb

msiobjput_srb ( msParam_t * inMSOPath,
                msParam_t * inCacheFilename,
                msParam_t * inFileSize )

Parameters:
[in] inMSOPath - a STR_MS_T containing the request sent to the external resource.
[in] inCacheFilename - a STR_MS_T containing the full path for the local cache filename to be written out.
[in] inFileSize - a STR_MS_T containing the size of the cache file.

Description:
This micro-service puts an object into a Storage Resource Broker data grid.

Note:
The inMSOPath starts with "srb:" The string after that has three parts separated by a colon, ":":
    The first part is the SRB host name.
    The second part is the SRB port number.
    The third part consists of two sections,
        A user name,
        An SRB logical pathname.
    The user name can have a zone name delineated by @.
    An example is:
        srb:srbbrick11.sdsc.edu:7676:testuser@sdsc:TESTUSER/UCHRI/home/srbAdmin.uchri/testdir/testFile

Example Usage:

myTestRule {
    # Input parameters are:
    # inMSOPath - the string sent to the remote Storage Resource Broker data grid
    # inCacheFilename - the full path of the cache file
    # inFileSize - the size of the cache file, found from ls on the vault
    # No output parameters
    msiobjput_srb(*Request, *Path, *Size);
}

INPUT *Request
"srb:srbbrick11.sdsc.edu:7676:testuser@sdsc:TESTUSER/UCHRI/home/srbAdmin.uchri/testdir/testFile",
*Path = "~/home/reagan/Vaulttesthome/rods/sub1/rodsfile", *Size = "15"
OUTPUT ruleExecOut
4.228  Module :: msoDrivers :: msiobjput_test

msiobjput_test ( msParam_t * inMSOPath,
msParam_t * inCacheFilename,
msParam_t * inFileSize )

Parameters:
[in]  inMSOPath      - a STR_MS_T containing the request sent to the external resource.
[in]  inCacheFilename - a STR_MS_T containing the full path for the local cache filename
[in]  inFileSize      - a STR_MS_T containing the size of the cache file.

Description:
This micro-service tests the micro-service object framework. It reads the specified file in the local vault (inCacheFilename), and writes the contents to the rodsLog file in the iRODS/server/log directory. The size of the file can be found by executing "ls –l" on the local cache.

Note:
The inMSOPath starts with "test:". This is followed with a test string. An example is:

        test:Test string

Example Usage:

myTestRule {
# Input parameters are:
#  inMSOPath  - the string that specifies a test of the micro-service object framework
#  inCacheFilename  - the full path of the cache file in the local storage vault
#  inFileSize  - the size of the cache file, found from a ls command on the storage vault
# Output from the rule is located in the rodsLog file:
#  MSO_TEST file contains: This is a test
        msiobjput_test(*Request, *Path, *Size);
}
INPUT *Request ="test:Test string", *Path = "/home/reagan/Vault/home/rods/rodsfile", *Size="15"
OUTPUT ruleExecOut
### 4.229 Module :: msoDrivers :: msiobjput_z3950

**msiobjput_z3950**

```
(msParam_t * inMSOPath,
msParam_t * inCacheFilename,
msParam_t * inFileSize)
```

**Parameters:**

- **[in]** `inMSOPath` - a STR_MS_T containing the request sent to the external resource.
- **[in]** `inCacheFilename` - a STR_MS_T containing the full path for the local cache filename to be written out.
- **[in]** `inFileSize` - a STR_MS_T containing the size of the cache file.

**Description:**

This micro-service puts data to a Z39.50 resource using micro-service drivers.

**Note:**

The `inMSOPath` starts with "z3950". The string after that is a Z39.50 compliant request string in either the MARC syntax or an XML syntax. Examples are:

- **MARC syntax:**
  
  `z3950:z3950.loc.gov:7090/Voyager?query=@attr 1=1003 Marx&recordsyntax=USMARC`

- **XML syntax:**
  
  `z3950:z3950.loc.gov:7090/Voyager?query=@attr 1=1003 Marx`

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  # inMSOPath - the string sent to the remote URL
  # inCacheFilename - the full path of the cache file
  # inFileSize - the size of the cache file, found from ls on the vault
  # No output parameters
  msiobjput_z3950(*Request, *Path, *Size);
}
```

INPUT

```
*Request = "z3950:z3950.loc.gov:7090/Voyager?query=@attr 1=1003 Marx&recordsyntax=USMARC",
*Path = "/home/reagan/Vaulttest/home/rods/sub1/rodsfile",
*Size = "15"
```

OUTPUT

```
ru
e
leExecOut
```
4.230 Module :: HDF :: msiH5Dataset_read

msiH5Dataset_read (msParam_t * inpH5DatasetParam,
                   msParam_t * outH5DatasetParam)

Parameters:
[in] inpH5DatasetParam - The input H5Dataset. Must be h5Dataset_MS_T.
[out] outH5DatasetParam - The output H5Dataset - Must be h5Dataset_MS_T.

Description:
This micro-service reads a dataset from an opened HDF5 file.

Note:
The input dataset name must be defined in the appropriate inpH5DatasetParam structure. The
inpH5DatasetParam structure is created by either HDF5 utilities, or HDF5 library calls, or the HDF5
viewer.

An application that generates the appropriate structure is available in:
iRODS/modules/hdf5/test/test_h5File.c
4.231 Module :: HDF :: msiH5Dataset_read_attribute

msiH5Dataset_read_attribute ( msParam_t * inpH5DatasetParam,
msParam_t * outH5DatasetParam )

Parameters:
[in] inpH5DatasetParam - The input H5Dataset. Must be h5Dataset_MS_T.
[out] outH5DatasetParam - The output H5Dataset - Must be h5Dataset_MS_T.

Description:
This micro-service is called internally by iRODS to read attributes of a dataset from an opened HDF5 file.

Note:
The input dataset name must be defined in the appropriate inpH5DatasetParam structure. The
inpH5DatasetParam structure is created by either HDF5 utilities, or HDF5 library calls, or the HDF5
viewer.

An application that generates the appropriate structure is available in:
  iRODS/modules/hdf5/test/test_h5File.c
4.232 Module :: HDF :: msiH5File_close

msiH5File_close (msParam_t * inpH5FileParam,
                 msParam_t * outH5FileParam)

Parameters:
[in] inpH5FileParam - The input H5File to close. Must be h5File_MS_T.
[out] outH5FileParam - the output H5File - Must be h5File_MS_T.

Description:
This micro-service closes an HDF5 file.

Note:
The input dataset name must be defined in the appropriate inpH5DatasetParam structure. The
inpH5DatasetParam structure is created by either HDF5 utilities, or HDF5 library calls, or the HDF5
viewer.

An application that generates the appropriate structure is available in:
   iRODS/modules/hdf5/test/test_h5File.c
4.233 Module :: HDF :: msiH5File_open

```
msiH5File_open (  msParam_t *  inpH5FileParam,
                 msParam_t *  inpFlagParam,
                 msParam_t *  outH5FileParam
              )
```

**Parameters:**
- **[in]** `inpH5FileParam` - The input H5File to open. Must be h5File_MS_T.
- **[in]** `inpFlagParam` - Input flag - INT_MS_T, that specifies the level of output detail.
- **[out]** `outH5FileParam` - the output H5File - Must be h5File_MS_T.

**Description:**
This micro-service opens an HDF5 file.

**Note:**
The input dataset name must be defined in the appropriate inpH5DatasetParam structure. The inpH5DatasetParam structure is created by either HDF5 utilities, or HDF5 library calls, or the HDF5 viewer.

An application that generates the appropriate structure is available in:

```
iRODS/modules/hdf5/test/test_h5File.c
```
4.234  Module :: HDF :: msiH5Group_read_attribute

**msiH5Group_read_attribute**  (  
  msParam_t* inpH5GroupParam,  
  msParam_t* outH5GroupParam  
)

**Parameters:**
- [in] inpH5GroupParam - The input H5Group - must be h5Dataset_MS_T.
- [out] outH5GroupParam - The output H5Group - must be h5Dataset_MS_T.

**Description:**
This micro-service reads attributes of a dataset from an opened H5Group.

**Note:**
The input dataset name must be defined in the appropriate inpH5DatasetParam structure. The inpH5DatasetParam structure is created by either HDF5 utilities, or HDF5 library calls, or the HDF5 viewer.

An application that generates the appropriate structure is available in:
- iRODS/modules/hdf5/test/test_h5File.c
4.235 Module :: Image :: msiImageConvert

msiImageConvert ( msParam_t * sourceParam,
                   msParam_t * sourceProp,
                   msParam_t * destParam,
                   msParam_t * destProp )

Parameters:

[in] sourceParam - the source file
[in] sourceProp - the source properties (Only image format is enabled).
[in] destParam - the destination file
[in] destProp - the destination properties (Only image format and/or compression type are enabled).

Description:
Read a source image file and write it out as a new image file in a chosen format.

Note:
The source and destination image files may be specified as:
- A string file path
- An integer file descriptor for an open file
- A data object
The destination file will be created if needed.
The source and destination files have optional property lists. The source property list may select the file format to use and which image in a file to read (if the file contains multiple images). The destination property list may select the file format to write and compression flags to use. The property is specified as a tagged string such as:

  <image.Format>gif</image.Format>

Queriable image properties are dependent on image format and may include:

<table>
<thead>
<tr>
<th>image.Colors</th>
<th>image.Depth</th>
<th>image.ResolutionUnit</th>
</tr>
</thead>
<tbody>
<tr>
<td>image.ColorSpace</td>
<td>image.Format</td>
<td>image.Rows</td>
</tr>
<tr>
<td>image.Columns</td>
<td>image.Gamma</td>
<td>image.XResolution</td>
</tr>
<tr>
<td>image.Compression</td>
<td>image.Interlace</td>
<td>image.YResolution</td>
</tr>
<tr>
<td>image.CompressionQuality</td>
<td>image.Orientation</td>
<td></td>
</tr>
</tbody>
</table>

Example Usage:

myTestRule { 
  # Input parameters:
  # srcFile - iRODS image file that will be converted
  # srcOptions - Optional srcFile property values
  #   (only image format values are allowed if this parameter is used)
  # destFile - iRODS image file that will be created to hold converted image
  # destOptions - Optional destFile properties string giving parameters for conversion
  #   (only file format and compression flags are allowed)
  #
  # Uses ImageMagick.
  #
  # Set properties for destOptions... in this case, the image.Format.
  #
  # NB: in this example, the file extension in the destFile is omitted
  # to illustrate the usage of destOptions.
# Get properties for srcFile for perusal
msiImageGetProperties(*srcFile, "null", *Prop);
writeLine("stdout", "Original file properties: ");
msiPrintKeyValPair("stdout", *Prop);

# Get image format just for checking
msiPropertiesGet(*Prop, "image.Format", *formatVal);
writeLine("stdout", "Original file format: ");
writeLine("stdout", ");

# Convert
msiImageConvert(*srcFile,"null","destFile","destOptions");

# Write a message to the server log
writeLine("serverLog", "Converting *srcFile to *destFile");

# Write a message to stdout
writeLine("stdout", "Converted file properties: ");
msiPrintKeyValPair("stdout", *Prop);
msiPropertiesGet(*Prop, "image.Format", *formatVal);
writeLine("stdout", "Format of converted file: ");

INPUT *srcFile="/tempZone/home/rods/image/ncdc.png", *destFile="/tempZone/home/rods/image/ncdc",
*destOptions="<image.Format>gif</image.Format>"
OUTPUT ruleExecOut

rulemsiImageConvert-compression.r
myTestRule {}
# Input parameters:
# srcFile - iRODS image file that will be converted
# srcOptions - Optional srcFile property values
# (only image format values are allowed if this parameter is used)
# destFile - iRODS image file that will be created to hold converted image
# destOptions - Optional destFile properties string giving parameters for conversion
# (only file format and compression flags are allowed)
#
# Uses ImageMagick.
#
# Set properties for destOptions... in this case, the compression
# Get properties for srcFile - just for perusing
msiImageGetProperties(*srcFile, "null", *Prop);
writeLine("stdout", "Original file properties: ");
msiPrintKeyValPair("stdout", *Prop);

# Get image format just for checking
msiPropertiesGet(*Prop, "image.Compression", *compVal);
writeLine("stdout", "Original compression: ");
writeLine("stdout", ");
# Convert
msiImageConvert(*srcFile,"null",*destFile,*destOptions);

# Write a message to the server log
writeLine("serverLog", "Converting *srcFile to *destFile");

# Write a message to stdout
writeLine("stdout", "");

# Write out properties of new image file
msiImageGetProperties(*destFile, "null", *Prop);
writeLine("stdout", "Converted file properties:");
msiPrintKeyValPair("stdout", *Prop);

msiPropertiesGet(*Prop, "image.Compression", *compVal);
writeLine("stdout", "");
writeLine("stdout", "Compression of converted file: *compVal");

} INPUT *srcFile="/tempZone/home/rods/image/ncdc.png", *destFile="/tempZone/home/rods/image/ncdc-recomp.png", *destOptions="<image.Compression>lossless</image.Compression>"
OUTPUT ruleExecOut

rulemsiImageConvert-no-properties.r
myTestRule {
  # Input parameters:
  # srcFile - iRODS image file that will be converted
  # destFile - iRODS image file that will be created and hold converted image
  #
  # Uses ImageMagick.
  #
  # Conversion is guided by file extensions in file names.
  #
  #
  # Call the microservice that converts the image from the type specified in
  # *srcFile into the type specified in *destFile

  # Write message to server log
  writeLine("serverLog", "Converting *srcFile to *destFile");

  # Write message to stdout
  writeLine("stdout", "Converting *srcFile to *destFile");
}

INPUT *srcFile="/tempZone/home/rods/image/ncdc.png", *destFile="/tempZone/home/rods/image/ncdc.gif"
OUTPUT ruleExecOut
Module :: Image :: msiImageGetProperties

```c
msiImageGetProperties( msParam_t * sourceParam,
                        msParam_t * sourceProp,
                        msParam_t * listParam )
```

**Parameters:**
- `[in]` `sourceParam` - the source file
- `[in]` `sourceProp` - the source properties
- `[out]` `listParam` - the returned properties list

**Description:**
Get the properties of an image file.

**Note:**
The source and destination image files may be specified as:
- A string file path
- An integer file descriptor for an open file
- A data object

Image properties are dependent on image type and may include:

<table>
<thead>
<tr>
<th>image.Colors</th>
<th>image.Depth</th>
<th>image.ResolutionUnit</th>
</tr>
</thead>
<tbody>
<tr>
<td>image.ColorSpace</td>
<td>image.Format</td>
<td>image.Rows</td>
</tr>
<tr>
<td>image.Columns</td>
<td>image.Gamma</td>
<td>image.XRresolution</td>
</tr>
<tr>
<td>image.Compression</td>
<td>image.Interlace</td>
<td>image.YResolution</td>
</tr>
<tr>
<td>image.CompressionQuality</td>
<td>image.Orientation</td>
<td></td>
</tr>
</tbody>
</table>

**Example Usage:**

```c
testRule {   
  // Input parameters:
  // inFile - input iRODS file (with complete path)
  // inOptions - list of options to query for
  // NB: The inOptions parameter seems to have no effect; all
  // properties are returned, independent of the value of inOptions.
  //
  // Output parameter:
  // outProperties - string containing returned properties list
  //
  // Call microservice to get the image properties
  msiImageGetProperties(*inFile, *inOptions, *outProperties);
  // Write message and all properties to stdout
  writeLine("stdout", "Getting properties of *inFile");
  writeLine("stdout", "");
  msiPrintKeyValPair("stdout", *outProperties);
  // Write message to server log
  writeLine("serverLog", "Getting properties of *inFile");
  // Write out value of a specific property
  msiPropertiesGet(*outProperties, "image.Compression", *val);
  writeLine("stdout", ");
  writeLine("stdout", "Value of image.Compression: *val");
}
```

INPUT *inFile="/tempZone/home/rods/image/ncdc.png", *inOptions="null"
OUTPUT ruleExecOut
4.237 Rule :: LibraryOfCongress :: rulegenerateBagIt.r

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

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

Example Usage:

```bash
generateBagIt {
# -------------------------------------
# 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
#
# # -------------------------------------
# ### - 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 = "NEWBAGITROOT" ++ "/data";
# *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);
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",*NEWBAGITROOT,"null",*Status);
### - gets filesize of new tarfile
msiSplitPath("NEWBAGITROOT.tar",*Coll,*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",*COLL,*TARFILE);
writeLine("stdout","To copy it to your local computer, use:");
writeLine("stdout","  iget -Pf *NEWBAGITROOT.tar *TARFILE");
writeLine("stdout",""");
###  - writes to rodsLog
  msiWriteRodsLog("BagIt bag created: *NEWBAGITROOT <- *BAGITDATA",*Status);
}
INPUT *BAGITDATA=$"/tempZone/home/rods/sub1",
*NEWBAGITROOT=$"/tempZone/home/rods/bagit"
OUTPUT ruleExecOut
Rule :: Integrity :: ruleintegrityACL.r

Description:
This rule verifies that for a designated user a specific ACL is present on each file in the collection.

Note:
Given a collection name, an iRODS user name, and an access permission, the rule loops over all files within the collection and verifies that the required access control has been set. The rule lists all files in the collection that do not have the required ACL. The access control names that can be tested are:

- null
- execute
- read annotation
- read system metadata
- read object
- write annotation
- create metadata
- modify metadata
- administer object
- create object
- modify object
- delete object
- create token
- delete token
- curate
- own

The example uses a micro-service from the ERA module, which must be enabled and compiled.

Example Usage:

```plaintext
integrityACL {
  # 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

  # Verify input path is a collection
  *Result = 1;
  msiIsColl(*Coll,*Result,*Status);
  if(*Result == 0) {
    writeLine("stdout","Input path *Coll is not a collection");
    fail;
  }

  # Get USER_ID for the input user name
  msiExecStrCondQuery("SELECT USER_ID where USER_NAME = '*User'",*GenQOut0);
  *Userid = ""
  foreach(*GenQOut0) {
    msiGetValByKey(*GenQOut0,"USER_ID", *Userid);
  }
  if(*Userid == "") {
    writeLine("stdout","Input user name *User is unknown");
    fail;
  }
}
```

# Get DATA_ACCESS_DATA_ID number that corresponds to requested access control
msiExecStrCondQuery("SELECT TOKEN_ID where TOKEN_NAMESPACE = 'access_type' and
TOKEN_NAME = '*Acl',*GenQOut1);
foreach(*GenQOut1)
  msiGetValByKey(*GenQOut1,"TOKEN_ID",*Access);
writeLine("stdout","Access control number of *Acl is *Access");

# Loop over files in the collection
msiMakeGenQuery("DATA_ID, DATA_NAME","COLL_NAME = '*Coll', *GenQInp);
msiExecGenQuery(*GenQInp, *GenQOut);
msiGetContInxFromGenQueryOut(*GenQOut,*ContInxNew);
while(*ContInxOld > 0)
  foreach(*GenQOut)
    msiGetValByKey(*GenQOut, "DATA_ID", *Dataid);
    msiGetValByKey(*GenQOut, "DATA_NAME", *File);
    msiMakeGenQuery("DATA_ACCESS_TYPE,DATA_ACCESS_USER_ID","DATA_ACCESS_DATA_ID = '*Dataid', *GenQm);
    msiExecGenQuery(*GenQm, *GenQOutm);
    msiGetValByKey(*GenQOutm, "DATA_ACCESS_USER_ID", *Userdid);
    if(*Userdid == *User)
      msiGetValByKey(*GenQOutm, "DATA_ACCESS_TYPE", *Datatype);
      if(*Datatype < *Access)
        writeLine("stdout","*File has wrong access permission, *Datatype");
    if(*Attrfound == 0)
      writeLine("stdout","*File is missing access controls for *User");
      *Count = *Count + 1;
  *ContInxOld = *ContInxNew;
  if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp,*GenQOut,*ContInxNew);}  
writeLine("stdout","Number of files in *Coll missing access control for *User is *Count");

INPUT *Coll = "/tempZone/home/rods", *User="rods", *Acl = "own"
OUTPUT ruleExecOut
4.239 Rule :: Integrity :: ruleintegrityAVU.r, ruleintegrityAVUvalue.r

Description:
This rule tests the files in a collection for the presence of a desired attribute name or the presence of both attribute name and attribute value.

Note:
The rule lists all files in a collection that do not have the input attribute name, or that do not have either the name or the desired attribute value. The latter case is useful to list the values present within an attribute name. The example uses a micro-service from the ERA module, which must be enabled and compiled.

Example Usage:

ruleintegrityAVU.r
integrityAVU {
# Input parameter is:
# Name of collection that will be checked
# Attribute name whose presence will be verified
# Output is:
# List of all files in the collection that are missing the attribute

# Verify that input path is a collection
msIsColl(*Coll,*Result, *Status);
if(*Result == 0) {
    writeLine("stdout","Input path *Coll is not a collection");
    fail;  }
*ContInxOld = 1;
*Count = 0;

# Loop over files in the collection
msiMakeGenQuery("DATA_ID,DATA_NAME","COLL_NAME = "*Coll", *GenQInp);
msiExecGenQuery(*GenQInp, *GenQOut);
msiGetContInxFromGenQueryOut(*GenQOut,*ContInxNew);
while(*ContInxOld > 0) {
    foreach(*GenQOut) {
        msiGetValByKey(*GenQOut,"DATA_ID", *Dataid);
        msiMakeGenQuery("META_DATA_ATTR_NAME","DATA_ID = "*Dataid", *GenQInp1);
        msiExecGenQuery(*GenQInp1, *GenQOut1);
        *Attrfound = 0;
        foreach(*GenQOut1) {
            msiGetValByKey(*GenQOut1,"META_DATA_ATTR_NAME",*Attrname);
            if(*Attrname == *Attr) {
                *Attrfound = 1;
            }
        }
        msiFreeBuffer(*GenQInp1);
        msiFreeBuffer(*GenQOut1);
        if(*Attrfound == 0) {
            msiGetValByKey(*GenQOut,"DATA_NAME", *File);
            writeLine("stdout","*File does not have attribute *Attr");
            *Count = *Count + 1;
        }
    }
    *ContInxOld = *ContInxNew;
    if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp,*GenQOut,*ContInxNew);}
}
}
writeLine("stdout","Number of files in *Coll missing attribute *Attr is *Count");
}
INPUT *Coll = "/tempZone/home/rods/sub1", *Attr = "DC.Relation"
OUTPUT ruleExecOut

ruleintegrityAVUvalue.r
integrityAVUvalue {
# Verify each file in a collection has a required attribute name and attribute value
# Input parameter is:
# Name of collection that will be checked
# Attribute name whose presence will be verified
# Attribute value that will be verified
# Output is:
# List of all files in the collection that are either missing the attribute
# of have the wrong attribute value

# Verify that input path is a collection
msIsColl(*Coll,*Result, *Status);
if(*Result == 0) {
    writeLine("stdout","Input path *Coll is not a collection");
    fail;  }
*ContInxOld = 1;
*Count = 0;

# Loop over files in the collection
msiMakeGenQuery("DATA_ID,DATA_NAME","COLL_NAME = '*Coll"*, *GenQInp);
msiExecGenQuery(*GenQInp, *GenQOut);
msiGetContInxFromGenQueryOut(*GenQOut,*ContInxNew);
while(*ContInxOld > 0) {
    foreach(*GenQOut) {
        msiGetValByKey(*GenQOut,"DATA_ID", *Dataid);
        msiGetValByKey(*GenQOut,"DATA_NAME", *File);
        msiMakeGenQuery("META_DATA_ATTR_NAME","DATA_ID = '*Dataid", *GenQInp1);
        msiExecGenQuery(*GenQInp1, *GenQOut1);
        *Attrfound = 0;
        foreach(*GenQOut1) {
            msiGetValByKey(*GenQOut1,"META_DATA_ATTR_NAME",*Attrname);
            if(*Attrname == *Attr) {
                *Attrfound = 1;
            }
        }
        msiMakeGenQuery("META_DATA_ATTR_VALUE","DATA_ID = '*Dataid' and
META_DATA_ATTR_NAME = '*Attrname",*GenQInp2);
        msiExecGenQuery(*GenQInp2,*GenQOut2);
        foreach(*GenQOut2) {
            msiGetValByKey(*GenQOut2,"META_DATA_ATTR_VALUE",*Attrval);
            if(*Attrval != "*Attrval") {writeLine("stdout","Incorrect attribute value for *Attr in *File in *Coll");}
        }
    }
    if(*Attrfound == 0) {
        writeLine("stdout","*File does not have attribute *Attr");
        *Count = *Count + 1;
    }
    *ContInxOld = *ContInxNew;
    if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp,*GenQOut,*ContInxNew);}
}
writeLine("stdout","Number of files in *Coll missing attribute *Attr is *Count");
}
INPUT *Coll = "/tempZone/home/rods/sub1", *Attr = "Event", *Attrv = "document"
OUTPUT ruleExecOut
4.240 Rule :: Integrity :: ruleintegrityExpiry.r

Description:
This rule checks whether files in a collection have expired or not expired.

Note:
Set the *Flag variable to "EXPIRED" or "NOT EXPIRED" to choose whether to list files that have exceeded their retention periods or that have not exceeded their retention period. The example uses a micro-service from the ERA module, which must be enabled and compiled.

Example Usage:

integrityExpiry {
    # Input parameter is:
    # Name of collection that will be checked
    # Flag for "EXPIRED" or for "NOT EXPIRED"
    # Output is:
    # List of all files in the collection that have either EXPIRED or NOT EXPIRED

    # Verify that input path is a collection
    msiIsColl(*Coll,*Result, *Status);
    if(*Result == 0) {
        writeLine("stdout","Input path *Coll is not a collection");
        fail;
    }
    *ContInxOld = 1;
    *Count = 0;
    *Counte = 0;
    msiGetIcatTime(*Time,"unix");

    # Loop over files in the collection
    msiMakeGenQuery("DATA_ID,DATA_NAME,DATA_EXPIRY","COLL_NAME = '*Coll'",
                    *GenQInp);
    msiExecGenQuery(*GenQInp, *GenQOut);
    msiGetContInxFromGenQueryOut(*GenQOut,*ContInxNew);
    while(*ContInxOld > 0) {
        foreach(*GenQOut) {
            msiGetValByKey(*GenQOut,"DATA_EXPIRY",*Attrname);
            if(*Attrname > *Time && *Flag == "NOT EXPIRED") {
                msiGetValByKey(*GenQOut,"DATA_NAME",*File);
                writeLine("stdout","File *File has not expired");
                *Count = *Count + 1;
            }
            if(*Attrname <= *Time && *Flag == "EXPIRED") {
                msiGetValByKey(*GenQOut,"DATA_NAME",*File);
                writeLine("stdout","File *File has expired");
                *Counte = *Counte + 1;
            }
        }
        *ContInxOld = *ContInxNew;
        if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp,*GenQOut,*ContInxNew);}
    }
    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 = "/tempZone/home/rods/sub1", *Flag = "EXPIRED"
OUTPUT ruleExecOut
4.241 Rule :: Integrity :: ruleintegrityFileSize.r

Description:
This rule checks to see if file sizes are NOT within a certain range.

Note:
The minimum allowed file size and maximum allowed file size are specified. A list of all files that do not
lie within this range is printed. The example uses a micro-service from the ERA module, which must be
enabled and compiled.

Example Usage:

integrityFileSize {
  # Input parameter is:
  # Name of collection that will be checked
  # Minimum file size allowed in collection
  # Maximum file size allowed in collection
  # Output is:
  # List of all files in the collection that have a size outside the allowed range

  # Verify that input path is a collection
  *Isizemax = int(*Sizemax);
  *Isizemin = int(*Sizemin);
  msiIsColl(*Coll,*Result, *Status);
  if(*Result == 0) {
    writeLine("stdout","Input path *Coll is not a collection");
    fail;
  }
  *ContInxOld = 1;
  *Count = 0;

  # Loop over files in the collection
  msiMakeGenQuery("DATA_ID,DATA_NAME,DATA_SIZE","COLL_NAME = '*Coll'", *GenQInp);
  msiExecGenQuery(*GenQInp, *GenQOut);
  msiGetContInxFromGenQueryOut(*GenQOut,*ContInxNew);
  while(*ContInxOld > 0) {
    foreach(*GenQOut) {
      msiGetValByKey(*GenQOut,"DATA_SIZE",*Dsize);
      *Idsize = int(*Dsize);
      if(*Idsize > *Isizemax) {
        msiGetValByKey(*GenQOut,"DATA_NAME","*File");
        writeLine("stdout","File *File with size *Dsize is larger than allowed size");
        *Count = *Count + 1;
      }
      if(*Idsize < *Isizemin) {
        msiGetValByKey(*GenQOut,"DATA_NAME","*File");
        writeLine("stdout","File *File with size *Dsize is smaller than allowed size");
        *Count = *Count + 1;
      }
    }
    *ContInxOld = *ContInxNew;
    if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp,*GenQOut, *ContInxNew);}
  }
  writeLine("stdout","Number of files in *Coll outside size range *Sizemin to *Sizemax is *Count");
}

INPUT *Coll = "/tempZone/home/rods/sub1", *Sizemin = "1000000", *Sizemax = "1000000000"
OUTPUT ruleExecOut
4.242  Rule :: Integrity :: ruleintegrityFileOwner.r

Description:
This rule checks if files in a given collection have a consistent owner.

Note:
The owner of each file is checked for compliance with a specified owner. All files with a different owner are listed. The owner is designated as the person who originally created the file. It is possible to give file ownership permission to other persons, but the original owner is not changed. The example uses a micro-service from the ERA module, which must be enabled and compiled.

Example Usage:

integrityFileOwner {
# Input parameter is:
#  Name of collection that will be checked
#  Owner name that will be verified
# Output is:
#  List of all files in the collection that have a different owner

# Verify that input path is a collection
msiIsColl(*Coll,*Result, *Status);
if(*Result == 0) {
    writeLine("stdout","Input path *Coll is not a collection");
    fail;
}
*ContInxOld = 1;
*Count = 0;

# Loop over files in the collection
msiMakeGenQuery("DATA_ID,DATA_NAME,DATA_OWNER_NAME","COLL_NAME = *Coll",*GenQInp);
msiExecGenQuery(*GenQInp, *GenQOut);
msiGetContInxFromGenQueryOut(*GenQOut,*ContInxNew);
while(*ContInxOld > 0) {
    foreach(*GenQOut) {
        msiGetValByKey(*GenQOut,"DATA_OWNER_NAME",*Attrname);
        if(*Attrname != *Attr) {
            msiGetValByKey(*GenQOut,"DATA_NAME",*File);
            writeLine("stdout","File *File has owner *Attrname");
            *Count = *Count + 1;
        }
    }
    *ContInxOld = *ContInxNew;
    if(*ContInxOld > 0) {msiGetMoreRows(*GenQInp,*GenQOut,*ContInxNew);}  
} writeLine("stdout","Number of files in *Coll with owner other than *Attr is *Count");
}
INPUT *Coll = "/tempZone/home/rods/sub1", *Attr = "rods"
OUTPUT ruleExecOut
4.243 Rule :: Example :: listMS.r

Description:
This rule lists all of the micro-services that have been enabled. This is listed for compatibility with iRODS version 2.5.

Note:
If the rule engine is unable to find a micro-service, the problem is most likely that the associated module has not been enabled and the micro-service has not been compiled. This is the same as rulemsiListEnabledMS.r

Example Usage:

ListMS {
    msiListEnabledMS(*KVPairs);
    writeKeyValPairs("stdout", *KVPairs, ": ");
}
INPUT null
OUTPUT ruleExecOut
4.244 Rule :: Example :: showCore.r

Description:
This rule lists all of the policies that have been loaded into the rule in-memory structure. This is listed for compatibility with iRODS version 2.5.

Note:
This is the same as rulesiAdmShowIRB.r.

Example Usage:

```
myTestRule {
    msiAdmShowIRB();
}
```

INPUT null
OUTPUT ruleExecOut
4.245  Module :: Properties :: msiPropertiesClear

msiPropertiesClear  (  msParam_t *  listParam  )

Parameters:
[in,out]  listParam   - a KeyValPair_MS_T, the property list to clear

Description:
Clear a property list

Example Usage:

myTestRule {
#  Input parameter is:
#  Property list

#  Create key-value string
msiString2KeyValPair(*Str,*KVpair);

#  Write out string
writeLine("stdout","Initial property list is");
msiPrintKeyValPair("stdout",*KVpair);

#  Clear properties list
msiPropertiesClear(*KVpair);

#  Verify property was cleared
writeLine("stdout","Changed property list is null");
msiPrintKeyValPair("stdout",*KVpair);
}
INPUT *Str="key1=value1"
OUTPUT ruleExecOut
4.246 Module :: Properties :: msiPropertiesClone

**msiPropertiesClone** (msParam_t * listParam,
msParam_t * cloneParam)

**Parameters:**
- [in] listParam - a KeyValPair_MS_T, the property list to clone
- [out] cloneParam - a KeyValPair_MS_T, the returned clone (new property list)

**Description:**
Clone a property list, returning a new property list

**Example Usage:**

```c
myTestRule {
    # Input parameter is:
    # Original property list
    # Output parameter is:
    # Cloned property list

    # Create key-value string
    msiString2KeyValPair(*Str,*KVpair);

    # Write out string
    writeLine("stdout","Initial property list is");
    msiPrintKeyValPair("stdout",*KVpair);

    # Clone the string
    msiPropertiesClone(*KVpair,*KVpair2);

    # Write out cloned string
    writeLine("stdout","Cloned property list is");
    msiPrintKeyValPair("stdout",*KVpair2);
}

INPUT *Str="key1=value1", *Keyword="key2", *Value="value2"
OUTPUT ruleExecOut
```
4.247  Module :: Properties :: msiPropertiesExists

msiPropertiesExists ( msParam_t * listParam,
                        msParam_t * keywordParam,
                        msParam_t * trueFalseParam )

Parameters:
[in,out] listParam - a KeyValPair_MS_T, the property list to look in
[in]   keywordParam - a STR_MS_T, a keyword to set
[out]  trueFalseParam - a INT_MS_T, true if set

Description:
Check for a property in a list. Return true (integer 1) if the keyword has a property value in the property list, and false (integer 0) otherwise. The property list is unmodified.

Example Usage:

myTestRule {
  # Input parameter
  # Property list
  # Keyword to find
  # Output parameter
  # Boolean result (1 if keyword is present)

  # Create key-value string
  msiString2KeyValPair(*Str,*KVpair);

  # Write out string
  writeLine("stdout","Initial property list is");
  msiPrintKeyValPair("stdout",*KVpair);

  # Verify property exists
  msiPropertiesExists(*KVpair,*Keyword,*Bvalue);
  writeLine("stdout","Property list checked for existence of *Keyword");
  writeString("stdout","Result is ");
  if(*Bvalue) {
    writeLine("stdout","Keyword *Keyword exists");
  }
  else {
    writeLine("stdout","Keyword *Keyword does not exist");
  }
}

INPUT *Str="key1=value1", *Keyword="key2", *Value="value2"
OUTPUT ruleExecOut
4.248  Module :: Properties :: msiPropertiesFromstring

**msiPropertiesFromstring**  (msParam_t * stringParam,

                  msParam_t * listParam)

**Parameters:**

- [in] stringParam - a STR_MS_T, a string buffer
- [out] listParam - a KeyValPair_MS_T, the property list with the strings added

**Description:**

Parse a string into a new property list. The existing property list, if any, is deleted.

**Note:**

The string format is:

```
<key1>value1</key1>,<key2>value2</key2>
```

**Example Usage:**

```c
myTestRule {
  # Input parameters are:
  # String of tagged values
  # Output parameter is:
  # Property list
  # Parse string into a property list
  msiPropertiesFromstring(*Str,*KVpair);
  writeLine("stdout","Property list is");
  msiPrintKeyValPair("stdout",*KVpair);
}
INPUT *Str="<key1>value1</key1>,<key2>value2</key2>"
OUTPUT ruleExecOut
```
Module :: Properties :: msiPropertiesGet

```c
msiPropertiesGet ( msParam_t * listParam,
                   msParam_t * keywordParam,
                   msParam_t * valueParam )
```

**Parameters:**
- `[in,out]` `listParam` - a KeyValPair_MS_T, the property list to be queried
- `[in]` `keywordParam` - a STR_MS_T, a keyword to get
- `[out]` `valueParam` - a STR_MS_T, the returned value

**Description:**
Get the value of a property in a property list. The property list is left unmodified.

**Example Usage:**
```c
myTestRule { 
  # Input parameters are:
  # Property list
  # Keyword to find
  # Output parameter is:
  # Value

  # Create key-value string
  msiString2KeyValPair(*Str,*KVpair);

  # Output property list
  writeLine("stdout","Property list is");
  msiPrintKeyValPair("stdout",*KVpair);

  # Extract property value
  msiPropertiesGet(*KVpair,*Str1,*Val);
  writeLine("stdout","Properties list keyword *Str1 has value *Val");
}
```

INPUT *Str="key1=value1", *Str1="key1"
OUTPUT ruleExecOut
4.250  Module :: Properties :: msiPropertiesNew

msiPropertiesNew  (  msParam_t *  listParam  )

Parameters:

  [out]  listParam  -  a KeyValPair_MS_T, the newly created property list

Description:
Create a new empty property list

Note:
A null property list is created.

Example Usage:

myTestRule {  
  # Output parameter is:  
  # Property list to create

  # Create new property list  
  msiPropertiesNew(*KVpair);

  # Add property to list  
  msiPropertiesAdd(*KVpair,*Keyword,*Value);

  # Write out property list  
  writeLine("stdout","Property list is");  
  msiPrintKeyValPair("stdout",*KVpair);
}  
INPUT *Keyword="key2", *Value="value2"  
OUTPUT ruleExecOut
4.251 Module :: Properties :: msiPropertiesRemove

**msiPropertiesRemove** (msParam_t* listParam,
                            msParam_t* keywordParam)

**Parameters:**
[in,out] listParam - a KeyValPair_MS_T, the property list that is being modified
[in] keywordParam - a STR_MS_T, a keyword to remove

**Description:**
Remove a property from a properties list.

**Example Usage:**

```c
myTestRule {
    # Input parameters are:
    # Properties list
    # Keyword to remove from list
    
    # Create properties list
    msiString2KeyValPair(*Str,*KVpair);
    
    # Add a property
    msiPropertiesAdd(*KVpair,*Keyword,*Value);
    
    # Write out property list
    writeLine("stdout","Property list is");
    msiPrintKeyValPair("stdout",*KVpair);
    
    # Remove a property
    msiPropertiesRemove(*KVpair,*Keyword);
    
    # Write out revised property list
    writeLine("stdout","Changed property list is");
    msiPrintKeyValPair("stdout",*KVpair);
}
```

INPUT *Str="key1=value1", *Keyword="key2", *Value="value2"
OUTPUT ruleExecOut
4.252 Module :: Properties :: msiPropertiesSet

msiPropertiesSet ( msParam_t * listParam,
                  msParam_t * keywordParam,
                  msParam_t * valueParam )

Parameters:
[in,out] listParam - a KeyValPair_MS_T, the property list that is being modified
[in]   keywordParam - a STR_MS_T, a keyword to set
[out]  valueParam   - a STR_MS_T, a value

Description:
Set the value of a property in a property list.

Note:
If the property is already in the list, its value is changed. Otherwise the property is added, similar to msiPropertiesAdd.

Example Usage:

myTestRule {  
    # Input parameters are:
    # Properties list
    # Keyword to check
    # Value to set
    
    # Create property string
    msiString2KeyValPair(*Str,*KVpair);
    msiPropertiesAdd(*KVpair,*Keyword,*Value);
    
    # Write property list
    writeLine("stdout","Property list is");
    msiPrintKeyValPair("stdout",*KVpair);
    
    # Change the value of a property in the list
    msiPropertiesSet(*KVpair,*Keyword,*Val2);
    
    # Write out new property list
    writeLine("stdout","Changed property list is");
    msiPrintKeyValPair("stdout",*KVpair);
    }

INPUT *Str="key1=value1", *Keyword="key2", *Value="value2", *Val2="newvalue2"
OUTPUT ruleExecOut
4.253  Module :: Properties :: msiPropertiesToString

msiPropertiesToString ( msParam_t * listParam,
                     msParam_t * stringParam )

Parameters:
[in]  listParam  - a KeyValPair_MS_T, the property list
[out] stringParam - a STR_MS_T, a string buffer

Description:
Convert a property list into a string buffer.

Note:
The property list is left unmodified. Requires that all properties have values.
Generates a string in the form:
    <key1>value1</key1>
    <key2>value2</key2>

Example Usage:

myTestRule {
    # Input parameters are:
    # Property list
    # Output parameter is:
    # String

    # Create property string
    msiString2KeyValPair(*Str,*KVpair);

    # Write out property list
    writeLine("stdout","Initial property list is");
    msiPrintKeyValPair("stdout",*KVpair);

    # Convert to a string
    msiPropertiesToString(*KVpair,*Strout);

    # Write out string
    writeLine("stdout","Generated string");
    writeLine("stdout","*Strout";)
}
INPUT *Str="key1=value1%key2=value2"
OUTPUT ruleExecOut
4.254  Module :: URL :: msiFtpGet

\texttt{msiFtpGet} (msParam_t * target, msParam_t * destObj, msParam_t * status)

**Parameters:**

- [in] \texttt{target} - Required - a STR_MS_T containing the remote URL.
- [in] \texttt{destObj} - Required - a DataObjInp_MS_T or a STR_MS_T which would be taken as the object's path.
- [out] \texttt{status} - a INT_MS_T containing the status.

**Description:**

This micro-service gets a remote file using FTP and writes it to an iRODS object.

**Note:**

This micro-service uses libcurl to open an ftp session with a remote server and read from a remote file. The results are written to a newly created iRODS object, one block at a time until the entire file is read.

**Example Usage:**

```plaintext
myTestRule {
    # Input parameters are:
    # Remote URL
    # New filepath within iRODS
    # Output parameter is:
    # Status
    msiFtpGet(*Target, *Destobj, *Status);
    writePosInt("stdout", *Status);
    writeLine("stdout", "");
}
```

```
INPUT *Target=\"ftp://mirror.nyi.net/apache/ant/README.html\",
*Destobj="/tempZone/home/rods/test/README.html"
```

```
OUTPUT ruleExecOut
```
4.255  Module :: URL :: msiTwitterPost

msiTwitterPost ( msParam_t * twittername,
msParam_t * twitterpass,
msParam_t * message,
msParam_t * status )

Parameters:
[in] twittername - Required - a STR_MS_T containing the twitter username.
[in] twitterpass - Required - a STR_MS_T containing the twitter password.
[in] message - Required - a STR_MS_T containing the message to post.
[out] status - An INT_MS_T containing the status.

Description:
Posts a message to twitter.com. A revised version may use OAuth to do the authentication.

Note:
This micro-service is deprecated, as Twitter has changed their authentication model.

This micro-service originally posted a message on twitter.com, aka a "tweet". A valid twitter account name and password must be provided. Special characters in the message can affect parsing of the POST form and create unexpected results. Avoid special characters if possible, or use quotes. This is intended for fun and for use in demos. Since your twitter password is passed unencrypted here, do not use this with a twitter account you do not wish to be compromised. Or if you do, change your password afterwards.

Example Usage:

myTestRule {
  # Input parameters are:
  # Twitter name
  # Twitter password
  # Message
  # Output parameter is:
  # Status
  #
  # This microservice worked with Twitter's basic authentication, through Spring 2010
  # - OAuth-based tweeting from iRODS is not yet implemented
  #
  msiTwitterPost(*Username, *Passwd, *Msg, *Status);
  writePosInt("stdout", *Status);
  writeLine("stdout", " is the status for twitter post");
}
INPUT *Username="rods", *Passwd="password", *Msg="Electronic Records Summer Camp is now open"
OUTPUT ruleExecOut
4.256 Module :: Web Services :: msiConvertCurrency

```c
msiConvertCurrency ( msParam_t * inConvertFromParam,
                    msParam_t * inConvertToParam,
                    msParam_t * outRateParam )
```

**Parameters:**
- `[in]` `inConvertFromParam` - a `msParam` of type `STR_MS_T`; 3-letter country code enumerated in structure `char *countryCodeNames[]`
- `[in]` `inConvertToParam` - a `msParam` of type `STR_MS_T`; 3-letter country code (same as above)
- `[out]` `outRateParam` - a `msParam` of operation status `STR_MS_T`; float number printed onto string

**Description:**
This micro-service returns conversion rates for currencies from one country to another, using web service provided by [http://www.webserviceX.NET](http://www.webserviceX.NET)

**Note:**
Web-service based micro-service to convert from one currency to another

**Example Usage:**

```plaintext
myTestRule {
    # Input parameters are:
    # Country code for original currency - 3 letters
    # Country code for desired currency - 3 letters
    # Output parameter is:
    # Conversion rate
    msiConvertCurrency(*InCurr,*OutCurr,*Rate);
    writeLine("stdout", "Conversion rate is *Rate");
}
```

**INPUT** `*InCurr="USA", *OutCurr="UK"`

**OUTPUT** `ruleExecOut`
4.257  Module :: Web Services :: msiGetQuote

msiGetQuote (  
msParam_t*  inSymbolParam,  
msParam_t*  outQuoteParam  )

Parameters:
[in]  inSymbolParam  - a msParam of type STR_MS_T which is a stock symbol.
[out]  outQuoteParam  - a msParam of type STR_MS_T which is a stock quotation converted from a 
float to a string

Description:
This micro-service returns stock quotation (delayed by web service) using web service provided by 
http://www.webserviceX.NET

Note:
Web service micro-service to get stock quotation

Example Usage:

myTestRule {  
# Input parameter is:
# Stock symbol
# Output parameter is:
# Stock quotation
  msiGetQuote(*Sym, *Quote);
  writeLine("stdout","For Stock *Sym the Quotation is *Quote");
}
INPUT *Sym = "ORCL"
OUTPUT ruleExecOut
4.258  Module :: Web Services :: msiIp2location

msiIp2location ( msParam_t * inIpParam,
                 msParam_t * inLocParam,
                 msParam_t * outLocParam )

Parameters:
[in]  inIpParam     - This msParam is of type STR_MS_T and inputs an ip-address.
[in]  inLocParam    - This msParam is of type STR_MS_T and is a license string provided
                    by http://ws.fraudlabs.com/
[out] outLocParam   - This msParam is of type STR_MS_T which is host location information

Description:
This micro-service returns host name and details given an IP address, using the web service provided by
http://ws.fraudlabs.com/. It consults a web service to convert an IP address to a location.

Note:
Web-service based micro-service for converting IP address to host name

Example Usage:

myTestRule {
  # Input parameters are:
  # IP address
  # License string from http://ws.fraudlabs.com/
  msiIp2location(*IpAddr,*License,*Loc);
  writeLine("stdout","Location is *Loc");
}
INPUT *IpAddr="132.249.32.95", *License="02-G34B-H86A"
OUTPUT ruleExecOut
4.259  Module :: Web Services :: msiObjByName

**msiObjByName** (  
msParam_t * inObjByNameParam,  
msParam_t * outRaParam,  
msParam_t * outDecParam,  
msParam_t * outTypParam  
)

**Parameters:**
- **[in]**  
inObjByNameParam - a msParam of type STR_MS_T which is an astronomical object name.
- **[out]**  
outRaParam - a msParam of type STR_MS_T which is a Right Ascension converted from a float to a string.
- **[out]**  
outDecParam - a msParam of type STR_MS_T which is a Declination converted from a float to a string.
- **[out]**  
outTypParam - a msParam of type STR_MS_T which is the type of object (e.g. star, galaxy,...).

**Description:**
This micro-service executes a web service to retrieve astronomy image by name. It returns position and type of an astronomical object given a name from the NASA/IPAC Extragalactic Database (NED) using web service at http://voservices.net/NED/ws_v2_0/NED.asmx.

**Note:**
Web-service based micro-service to get type and position of an astronomical object in NED database given a name.

**Example Usage:**

```c
myTestRule {  
  # Input parameter is:  
  # Astronomical object name  
  # Output parameters are:  
  # Right Ascension  
  # Declination  
  # Type of object  
  msiObjByName(*objName,*RA,*DEC,*TYPE);  
  writeLine("stdout","Right ascension is *RA, declination is *DEC, type is *Type");  
}
INPUT *objName=$"m100"
OUTPUT ruleExecOut
```
4.260  Module :: Web Services :: msiSdssImgCutout_GetJpeg

msiSdssImgCutout_GetJpeg (     msParam_t *  inRaParam,
                              msParam_t *  inDecParam,
                              msParam_t *  inScaleParam,
                              msParam_t *  inWidthParam,
                              msParam_t *  inHeightParam,
                              msParam_t *  inOptParam,
                              msParam_t *  outImgParam  )

Parameters:
[in]  inRaParam        - a msParam of type STR_MS_T which is a Right Ascension converted from
                             a float to a string.
[in]  inDecParam       - a msParam of type STR_MS_T which is a Declination converted from
                             a float to a string.
[in]  inScaleParam     - a msParam of type STR_MS_T which is a Scaling factor converted from
                             a float to a string.
[in]  inWidthParam     - a msParam of type STR_MS_T which is a Width of image converted from a
                             float to a string.
[in]  inHeightParam    - a msParam of type STR_MS_T which is a height of image converted from
                             a float to a string.
[in]  inOptParam       - a msParam of type STR_MS_T which is other optional
                             parameters.
[out] outImgParam      - a msParam of type BUF_LEN_MS_T which is the Image Buffer.

Description:
This micro-service returns an image buffer given a position and cutout from the SDSS Image Cut Out
service using web service provided by http://skyserver.sdss.org

Example Usage:

myTestRule {
  # Input parameters are:
  # Right Ascension
  # Declination
  # Scaling factor
  # Image width
  # Image height
  # Optional parameter for SDSS web service
  # Output parameter is:
  # Image buffer
}
INPUT *RA=$"185.72", *DEC=$"15.82", *Scale=$"0.396127", *Width=$"64", *Height=$"64",
*Opt=$"GPST"
OUTPUT ruleExecOut
Module :: XML :: msiLoadMetadataFromXml

```c
msiLoadMetadataFromXml ( msParam_t * targetObj,
                         msParam_t * xmlObj )
```

**Parameters:**
- `[in]` targetObj - Optional - a msParam of type DataObjInp_MS_T or STR_MS_T that specified the file on which the metadata is registered.
- `[in]` xmlObj - a msParam of type DataObjInp_MS_T or STR_MS_T that names the metadata source file.

**Description:**
This micro-service parses an XML iRODS file to extract metadata tags.

**Note:**
A sample-processed.xml file showing the expected AVU format.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
  <AVU>
    <Target>/tempZone/home/rods/XML/sample.xml</Target>
    <Attribute>Order ID</Attribute>
    <Value>889923</Value>
    <Unit />
  </AVU>
  <AVU>
    <Target>/tempZone/home/rods/XML/sample.xml</Target>
    <Attribute>Order Person</Attribute>
    <Value>John Smith</Value>
    <Unit />
  </AVU>
</metadata>
```

**Example Usage:**

```plaintext
myTestRule {
  # Input parameters are:
  # 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 microservice requires libxml2.
  # call the microservice
  msiLoadMetadataFromXml(*targetObj, *xmlObj);
  # write message to the log file
  writeLine("serverLog","Extracted metadata from *xmlObj and attached to *targetObj");
  # write message to stdout
  writeLine"stdout","Extracted metadata from *xmlObj and attached to *targetObj");
}
```

INPUT *xmlObj="/tempZone/home/rods/XML/sample-processed.xml", *targetObj=""
OUTPUT ruleExecOut
4.262 Module :: XML :: msiXmlDocSchemaValidate

msiXmlDocSchemaValidate ( msParam_t * xmlObj, msParam_t * xsdObj, msParam_t * status )

Parameters:
[in] xmlObj - a msParam of type DataObjInp_MS_T or STR_MS_T which is an iRODS path for the XML object.
[in] xsdObj - a msParam of type DataObjInp_MS_T or STR_MS_T which is iRODS path for the XSD object.
[out] status - a msParam of type INT_MS_T which is a validation result.

Description:
This micro-service validates an XML file against an XSD schema, both stored as iRODS objects.

Note:
Sample XML file:

```
<shiporder orderid="889923"
xmni:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
    <orderperson>John Smith</orderperson>
    <shipto>
        <name>Ola Nordmann</name>
        <address>Langgt 23</address>
        <city>4000 Stavanger</city>
        <country>Norway</country>
    </shipto>
    <item>
        <title>Empire Burlesque</title>
        <note>Special Edition</note>
        <quantity>1</quantity>
        <price>10.90</price>
    </item>
    <item>
        <title>Hide your heart</title>
        <quantity>1</quantity>
        <price>9.90</price>
    </item>
</shiporder>
```

Sample schema file:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="stringtype">
        <xs:restriction base="xs:string"/>
    </xs:simpleType>
    <xs:simpleType name="inttype">
        <xs:restriction base="xs:positiveInteger"/>
    </xs:simpleType>
    <xs:simpleType name="dectype">
        <xs:restriction base="xs:decimal"/>
    </xs:simpleType>
</xs:schema>
```
<xs:simpleType name="orderidtype">
    <xs:restriction base="xs:string">
        <xs:pattern value="[0-9]{6}"/>
    </xs:restriction>
</xs:simpleType>

<xs:complexType name="shiptotype">
    <xs:sequence>
        <xs:element name="name" type="stringtype"/>
        <xs:element name="address" type="stringtype"/>
        <xs:element name="city" type="stringtype"/>
        <xs:element name="country" type="stringtype"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="itemtype">
    <xs:sequence>
        <xs:element name="title" type="stringtype"/>
        <xs:element name="note" type="stringtype" minOccurs="0"/>
        <xs:element name="quantity" type="inttype"/>
        <xs:element name="price" type="dectype"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="shipordertype">
    <xs:sequence>
        <xs:element name="orderperson" type="stringtype"/>
        <xs:element name="shipto" type="shiptotype"/>
        <xs:element name="item" maxOccurs="unbounded" type="itemtype"/>
    </xs:sequence>
    <xs:attribute name="orderid" type="orderidtype" use="required"/>
</xs:complexType>

<xs:element name="shiporder" type="shipordertype"/>
</xs:schema>

Sample XSLT template file:

------------------------------
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/shiporder">
        <metadata>
            <AVU>
                <Attribute>Order ID</Attribute>
                <Value><xsl:value-of select="@orderid"></Value>
                <Unit />
            </AVU>
            <AVU>
                <Attribute>Order Person</Attribute>
                <Value><xsl:value-of select="orderperson"></Value>
                <Unit />
            </AVU>
            <AVU>
                <Attribute>Ship To</Attribute>
                <Value><xsl:value-of select="shipto/name"></Value>
                <Unit />
            </AVU>
        </metadata>
    </xsl:template>
</xsl:transform>
Sample transformed file:

--------------------------------
</xsl:template>
</xsl:transform>
<?xml version="1.0" encoding="UTF-8"?>
<metadata>
  <AVU>
    <Attribute>Order ID</Attribute>
    <Value>889923</Value>
    <Unit />
  </AVU>
  <AVU>
    <Attribute>Order Person</Attribute>
    <Value>John Smith</Value>
    <Unit />
  </AVU>
  <AVU>
    <Attribute>Ship To</Attribute>
    <Value>Ola Nordmann</Value>
    <Unit />
  </AVU>
</metadata>

Example Usage:

myTestRule {
  # Input parameters:
  # xmlObj       - XML file (an iRODS object)
  #xsdObj       - XSD schema file (an iRODS object)
  # Output parameter:
  # Status - integer indicating success of failure of validation
  # (0) on success
  #
  # This microservice requires libxml2

  # call the microservice
  msiXmlDocSchemaValidate(*xmlObj, *xsdObj, *Status);

  # write information to stdout
  writeLine("stdout","Validated *xmlObj against *xsdObj");

  # write integer into stdout
  writePosInt("stdout",*Status);
  writeLine("stdout","\n");
}
INPUT *xmlObj="/tempZone/home/rods/XML/sample.xml",
*xsdObj="/tempZone/home/rods/XML/sample.xsd"
OUTPUT ruleExecOut
**4.263 Module :: XML :: msiXsltApply**

`msiXsltApply (msParam_t * xsltObj,
               msParam_t * xmlObj,
               msParam_t * msParamOut)`

**Parameters:**
- [in] `xsltObj` - a msParam of type DataObjInp_MS_T or STR_MS_T for the stylesheet
- [in] `xmlObj` - a msParam of type DataObjInp_MS_T or STR_MS_T for the XML file
- [out] `msParamOut` - a msParam of operation status BUF_LEN_MS_T

**Description:**
This function applies an XSL stylesheet to an XML file, both existing iRODS objects.

**Note:**

Sample XML file
-------------
```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

Sample XSL file
-------------
```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
  <xsl:template match="/shiporder">
    <metadata>
      <AVU>
        <Attribute>Order ID</Attribute>
        <Value><xsl:value-of select="@orderid"/></Value>
      </AVU>
    </metadata>
    <AVU>
      <Attribute>Order Person</Attribute>
    </AVU>
  </xsl:template>
</xsl:transform>
```
Example Usage:

myTestRule {
    # Transform an XML file according to a provided XSL stylesheet
    # This rule calls microservice xsltApplyStylesheet from libsxlt to do the transformation
    # Typical usage:
    # Transform an XML file to the AVU format required for microservice msiLoadMetadataFromXML
    # Input parameters:
    # xsltObj - the XSL stylesheet file (an iRODS object)
    # xmlObj - the input XML file (an iRODS object)
    # note input parameters use the full iRODS path names
    # Output parameters:
    # outBuf - buffer containing the transformed XML data
    #
    # call the micro-service
    msiXsltApply(*xsltObj, *xmlObj, *outBuf);

    # write the output buffer to stdout
    writeBytesBuf("stdout", *outBuf);

    # write output file
    msiDataObjCreate(*Path,*OFlag,*D_FD);
    msiDataObjWrite(*D_FD,*outBuf,*W_len);
    msiDataObjClose(*D_FD,*Status1);
}

INPUT *xmlObj="/tempZone/home/rods/XML/sample.xml", *xsltObj="/tempZone/home/rods/XML/sample.xsl", *Path="/tempZone/home/rods/XML/sample-processed.xml", *OFlag="destRescName=demoResc++++forceFlag=
OUTPUT ruleExecOut
Module :: Z3950 :: msiz3950Submit

msiz3950Submit (msParam_t * serverName,
               msParam_t * query,
               msParam_t * recordSyntax,
               msParam_t * outParam)

Parameters:
[in] serverName - a STR_MS_T containing the name of the Z39.50 server
[in] query - a STR_MS_T containing the input query to the Z39.50 server
[in] recordSyntax - a STR_MS_T containing the preferred syntax for the returned record
[out] outParam - a STR_MS_T containing the retrieved record

Description:
Retrieves a record from a Z39.50 server

Example Usage:

myTestRule {
  # Input parameters are:
  # Z30.50 server name
  # Query
  # Syntax for returned record
  # Output parameter is:
  # Result
  msiz3950Submit(*Server,*Query,*RecordSyntax,*Out);
  writeBytesBuf("stdout", *Out);
}
INPUT *Server="z3950.loc.gov:7090/Voyager", *Query="@attr 1=1003 Marx",
*RecordSyntax="USMARC"
OUTPUT ruleExecOut
APPENDIX A: rodsPackTable.h DATA STRUCTURE TYPES

The following data structure types are supported in iRODS for a variety of purposes, including passing parameters for rule invocation using the "msParam" structure and for passing values between client-server and server-server interactions. The values can be found in the file "~/iRODS/lib/core/include/rodsPackTable.h".

For each data structure type, the name of an associated packing instruction is defined that specifies the elements that comprise the data structure. The packing instructions are listed in Appendix C.

```c
#ifndef RODS_PACK_TABLE_H
#define RODS_PACK_TABLE_H

#include "rods.h"
#include "packStruct.h"
#include "rodsPackInstruction.h"
#include "rodsGenQuery.h"
#include "reGlobalsExtern.h"
#include "apiHeaderAll.h"

#define UNKNOWN_SIZE -1

packType_t packTypeTable[] = {
    {"char", PACK_CHAR_TYPE, sizeof (char)},
    {"bin", PACK_BIN_TYPE, sizeof (char)},
    {"str", PACK_STR_TYPE, sizeof (char)},
    {"piStr", PACK_PI_STR_TYPE, sizeof (char)}, /* str containing pi */
    {"int", PACK_INT_TYPE, 4},
    {"double", PACK_DOUBLE_TYPE, 8},
    {"struct", PACK_STRUCT_TYPE, UNKNOWN_SIZE},
    {"?", PACK_DEPENDENT_TYPE, UNKNOWN_SIZE},
    {"%", PACK_INT_DEPENDENT_TYPE, UNKNOWN_SIZE},
};

int NumOfPackTypes = (sizeof (packTypeTable) / sizeof (packType_t));

packConstantArray_t PackConstantTable[] = {
    {"HEADER_TYPE_LEN", HEADER_TYPE_LEN},
    {"NAME_LEN", NAME_LEN},
    {"LONG_NAME_LEN", LONG_NAME_LEN},
    {"MAX_NAME_LEN", MAX_NAME_LEN},
    {"SHORT_STR_LEN", SHORT_STR_LEN},
    {"TIME_LEN", TIME_LEN},
    {"DIR_LEN", DIR_LEN},
    {"ERR_MSG_LEN", ERR_MSG_LEN},
    {"MAX_SQL_ATTR", MAX_SQL_ATTR},
    {"RULE_SET_DEF_LENGTH", RULE_SET_DEF_LENGTH},
    {"META_STR_LEN", META_STR_LEN},
    {"CHALLENGE_LEN", CHALLENGE_LEN},
    {"RESPONSE_LEN", RESPONSE_LEN},
    {"MAX_PASSWORD_LEN", MAX_PASSWORD_LEN}, /* HDF5 constant */
    {"OBJID_DIM", 2},
    {"H5S_MAX_RANK", 32},
    {"H5DATASPACE_MAX_RANK", 32},
    {"HUGE_NAME_LEN", HUGE_NAME_LEN}, /* end of HDF5 */
};

packInstructArray_t RodsPackTable[] = {
    {"STR_PI", STR_PI},
    {"IRODS_STR_PI", IRODS_STR_PI},
    {"STR_PTR_PI", STR_PTR_PI},
    {"INT_PI", INT_PI},
    {"BUF_LEN_PI", BUF_LEN_PI},
    {"MsgHeader_PI", MsgHeader_PI},
    {"StartupPack_PI", StartupPack_PI},
};
```
APPENDIX B: LIST OF msParam.c STRUCTURES

The "msParam.h" file contains the structure definitions for the "msParam" structure and other structures. Definitions for the parameters used within the iRODS software are listed. The utility functions that manipulate the "msParam" structure are defined. The header information is listed below. This file is available in the iRODS source release at "~/iRODS/lib/core/include/msParam.h". For each input and output parameter used by a micro-service, a data structure type is defined. For the specified data structure type, a name of a packing instruction (PI) is defined below that defines the elements of the structure. In Appendix C, the packing instructions are listed.

```c
#define msParam_t struct MsParam {
    char *label; /* this is the name of the packing instruction in * rodsPackTable.h */
    char *type;
    void *inOutStruct;
    bytesBuf_t *inpOutBuf;
} msParam_t;
```

```c
typedef struct MsParamArray {
    int len;
```
int oprType;
msParam_t **msParam;
}
msParamArray_t;

#define MS_INP_SEP_STR "++++" /* the separator str for msInp */
#define MS_NULL_STR "null" /* no input */
typedef struct ParsedMsKeyValStr {
char *inpStr;
char *endPtr; /* end pointer */
char *curPtr; /* current position */
char *kwPtr;
char *valPtr;
} parsedMsKeyValStr_t;
typedef struct ValidKeyWd {
int flag;
char *keyWd;
} validKeyWd_t;

/* valid keyWd flags for dataObjInp_t */
#define RESC_NAME_FLAG 0x1
#define DEST_RESC_NAME_FLAG 0x2
#define BACKUP_RESC_NAME_FLAG 0x4
#define FORCE_FLAG_FLAG 0x8
#define ALL_FLAG 0x10
#define LOCAL_PATH_FLAG 0x20
#define VERIFY_CHKSUM_FLAG 0x40
#define IRODS_ADMIN_FLAG 0x80
#define UPDATE_REPL_FLAG 0x100
#define REPL_NUM_FLAG 0x200
#define DATA_TYPE_FLAG 0x400
#define CHKSUM_ALL_FLAG 0x800
#define FORCE_CHKSUM_FLAG 0x1000
#define FILE_PATH_FLAG 0x2000
#define CREATE_MODE_FLAG 0x4000
#define OPEN_FLAGS_FLAG 0x8000
#define COLL_FLAGS_FLAG 0x8000
#define DATA_SIZE_FLAGS 0x10000
#define NUM_THREADS_FLAG 0x20000
#define OPR_TYPE_FLAG 0x40000
#define OBJ_PATH_FLAG 0x80000
#define COLL_NAME_FLAG 0x80000
#define IRODS_RMTRASH_FLAG 0x100000
#define IRODS_ADMIN_RMTRASH_FLAG 0x200000
#define DEF_RESC_NAME_FLAG 0x400000
#define RBUDP_TRANSFER_FLAG 0x800000
#define RBUDP_SEND_RATE_FLAG 0x1000000
#define RBUDP_PACK_SIZE_FLAG 0x2000000
#define BULK_OPR_FLAG 0x4000000

int resetMsParam (msParam_t *msParam);
int clearMsParam (msParam_t *msParam, int freeStruct);
int addMsParam (msParamArray_t *msParamArray, char *label, char *packInstruct, void *inOutStruct, bytesBuf_t *inpOutBuf);
int addIntParamToArray (msParamArray_t *msParamArray, char *label, int inpInt);
int addMsParamToArray (msParamArray_t *msParamArray, char *label, char *type, void *inOutStruct, bytesBuf_t *inpOutBuf, int replFlag);
int replMsParamArray (msParamArray_t *msParamArray, msParamArray_t *outMsParamArray);
int replMsParam (msParam_t *msParam, msParam_t *outMsParam);
int replInOutStruct (void *inStruct, void **outStruct, char *type);
int fillMsParam (msParam_t *msParam, char *label, char *type, void *inOutStruct, bytesBuf_t *inpOutBuf);
int fillBufLenInMsParam (msParam_t *msParam, int myInt, bytesBuf_t *bytesBuf);
int parseMspForDataObjInp (msParam_t *inpParam, dataObjInp_t *dataObjInpCache, dataObjInp_t **outDataObjInp, int writeToCache);
int parseMspForCollInp (msParam_t *inpParam, collInp_t *collInpCache, collInp_t **outCollInp, int writeToCache);
int parseMspForCondInp (msParam_t *inpParam, keyValPair_t *condInput, char *condKw);
int parseMspForCondKW (msParam_t *inpParam, keyValPair_t *condInput);
int parseMspForPhyPathReg (msParam_t *inpParam, keyValPair_t *condInput);
int parseMspForPosInt (msParam_t *inpParam);
char *parseMspForStr (msParam_t *inpParam);
int parseMspForDataObjCopyInp (msParam_t *inpParam, dataObjCopyInp_t *dataObjCopyInpCache, dataObjCopyInp_t **outDataObjCopyInp);
int parseMspForExecCmdInp (msParam_t *inpParam, execCmd_t *execCmdInpCache, execCmd_t **outExecCmdInp);
void *getMspInOutStructByLabel (msParamArray_t *msParamArray, char *label);
int getStdoutInExecCmdOut (msParam_t *inpExecCmdOut, char **outStr);
int getStderrInExecCmdOut (msParam_t *inpExecCmdOut, char **outStr);
int initParsedMsKeyValStr (char *inpStr, parsedMsKeyValStr_t *parsedMsKeyValStr);
int clearParsedMsKeyValStr (parsedMsKeyValStr_t *parsedMsKeyValStr);
int getNextKeyValFromMsKeyValStr (parsedMsKeyValStr_t *parsedMsKeyValStr);
int parseMsKeyValStrForDataObjInp (msParam_t *inpParam, dataObjInp_t *dataObjInp, char *hintForMissingKw, int validKwFlags, char **outBadKeyWd);
int chkDataObjInpKw (char *keyWd, int validKwFlags);
int parseMsKeyValStrForCollInp (msParam_t *inpParam, collInp_t *collInp, char *hintForMissingKw, int validKwFlags, char **outBadKeyWd);
int chkCollInpKw (char *keyWd, int validKwFlags);
int addKeyValToMspStr (msParam_t *keyStr, msParam_t *valStr, msParam_t *msKeyValStr);
int chkStructFileExtAndRegInpKw (char *keyWd, int validKwFlags);
int parseMsKeyValStrForStructFileExtAndRegInp (msParam_t *inpParam, structFileExtAndRegInp_t *structFileExtAndRegInp, char *hintForMissingKw, int validKwFlags, char **outBadKeyWd);

#endif  __cplusplus
}
APPENDIX C: LIST OF rodsPackInstruct.h PACKING INSTRUCTION DEFINITIONS

The structures used as input and output parameters in micro-services are listed below. There are a total of 74 data structures that are currently used to manage exchange of structured information in the data management workflows. Given the name of the data structure, the associated packing instruction can be found from Appendix A or Appendix B. The actual data structure that is used by the micro-service can then be found from the following list.

```c
#ifndef PACK_INSTRUCT_H
#define PACK_INSTRUCT_H

#define IRODS_STR_PI "str myStr[MAX_NAME_LEN];"
#define STR_PI "str myStr;"
#define STR_PTR_PI "str *myStr;"
#define PI_STR_PI "piStr myStr[MAX_NAME_LEN];"
#define INT_PI "int myInt;"
#define DOUBLE_PI "double myDouble;"

/* packInstruct for msgHeader_t */
#define MsgHeader_PI "str type[HEADER_TYPE_LEN]; int msgLen; int errorLen; int bsLen; int info;

/* packInstruct for startupPack_t */
#define StartupPack_PI "int irodsProt; int reconnFlag; int connectCnt; str proxyUser[NAME_LEN]; str proxyRcatZone[NAME_LEN]; str clientUser[NAME_LEN]; str clientRcatZone[NAME_LEN]; str relVersion[NAME_LEN]; str apiVersion[NAME_LEN]; str option[NAME_LEN];"

/* packInstruct for version_t */
#define Version_PI "int status; str relVersion[NAME_LEN]; str apiVersion[NAME_LEN]; int reconnPort; str reconnAddr[LONG_NAME_LEN]; int cookie;"

/* packInstruct for rErrMsg_t */
#define RErrMsg_PI "int status; str msg[ERR_MSG_LEN];"

/* packInstruct for rError_t */
#define RError_PI "int count; struct *RErrMsg_PI[count];"

#define RHostAddr_PI "str hostAddr[LONG_NAME_LEN]; str rodsZone[NAME_LEN]; int port; int dummyInt;"

#define RODS_STAT_T_PI "double st_size; int st_dev; int st_ino; int st_mode; int st_nlink; int st_uid; int st_gid; int st_rdev; int st_atim; int st_mtim; int st_ctim; int st_blksize; int st_blocks;"

#define RODS_DIRENT_T_PI "int d_offset; int d_ino; int d_reclen; int d_namlen; str d_name[DIR_LEN];"

#define KeyValPair_PI "int ssLen; str *keyWord[ssLen]; str *svalue[ssLen];"

#define InxIvalPair_PI "int iiLen; int *inx(iiLen); int *ivalue(iiLen);"

#define InxValPair_PI "int isLen; int *inx(isLen); str *svalue[isLen];"

#define DataObjInp_PI "str objPath[MAX_NAME_LEN]; int createMode; int openFlags; double offset; double dataSize; int numThreads; int oprType; struct *SpecColl_PI; struct KeyValPair_PI;"

#define OpenedDataObjInp_PI "int l1descInx; int len; int whence; int oprType; double offset; double bytesWritten; struct KeyValPair_PI;"

#define PortalOprOut_PI "int status; int l1descInx; int numThreads; str checksum[NAME_LEN]; struct PortalList_PI;"
```

359
```c
#define DataOprInp_PI "int oprType; int numThreads; int srcL3descInx; int destL3descInx; int srcRescTypeInx; int destRescTypeInx; double offset; double dataSize; struct KeyValPair_PI;"

#define CollInpNew_PI "str collName[MAX_NAME_LEN]; int flags; int oprType; struct KeyValPair_PI;"

#ifndef COMPAT_201
#define CollInp_PI "str collName[MAX_NAME_LEN]; struct KeyValPair_PI;"
#endif

#define GenQueryInp_PI "int maxRows; int continueInx; int partialStartIndex; int options; struct KeyValPair_PI; struct InxIvalPair_PI; struct InxValPair_PI;"

#define SqlResult_PI "int attriInx; int reslen; str *value(rowCnt)(reslen);"

#define GenQueryOut_PI "int rowCnt; int attriCnt; int continueInx; int totalRowCount; struct SqlResult_PI[MAX_SQL_ATTR];"

#define GenArraysInp_PI "int rowCnt; int attriCnt; int continueInx; int totalRowCount; struct KeyValPair_PI; struct SqlResult_PI[MAX_SQL_ATTR];"
#define DataObjInfo_PI "str objPath[MAX_NAME_LEN]; str rescName[NAME_LEN]; str rescGroupName[NAME_LEN]; str dataType[NAME_LEN]; double dataSize; str chksum[NAME_LEN]; str version[NAME_LEN]; str filePath[MAX_NAME_LEN]; str *rescInfo; str dataOwnerName[NAME_LEN]; str dataOwnerZone[NAME_LEN]; int replNum; int replStatus; str statusString[NAME_LEN]; double dataId; double collId; int dataMapId; int flags; str dataComments[LONG_NAME_LEN]; str dataExpiry[TIME_LEN]; str dataCreate[TIME_LEN]; str dataModify[TIME_LEN]; str dataAccess[NAME_LEN]; int dataAccessInx; int writeFlag; str destRescName[NAME_LEN]; str backupRescName[NAME_LEN]; str subPath[MAX_NAME_LEN]; int *specColl; int *next;"

/* transStat_t is being replaced by transferStat_t because of the 64 bits * padding */
#define TransStat_PI "int numThreads; double bytesWritten;"
#define TransferStat_PI "int numThreads; int flags; double bytesWritten;"
#define RescGrpInfo_PI "str rescGroupName[NAME_LEN]; str *rescName; int status; int dummy; int *cacheNext; struct *RescGrpInfo_PI;"
#define AuthInfo_PI "str authScheme[NAME_LEN]; int authFlag; int flag; int ppid; str host[NAME_LEN]; str authStr[NAME_LEN];"
#define UserOtherInfo_PI "str userInfo[NAME_LEN]; str userComments[NAME_LEN]; str userCreate[TIME_LEN]; str userModify[TIME_LEN];"
#define UserInfo_PI "str userName[NAME_LEN]; str rodsZone[NAME_LEN]; str userType[NAME_LEN]; int sysUid; struct AuthInfo_PI; struct UserOtherInfo_PI;"
#define CollInfo_PI "double collId; str collName[MAX_NAME_LEN]; str collParentName[MAX_NAME_LEN]; str collOwnerName[NAME_LEN]; str collOwnerZone[NAME_LEN]; int collMapId; int collAccessInx; str collComments[LONG_NAME_LEN]; str collInheritance[LONG_NAME_LEN]; str collExpiry[TIME_LEN]; str collCreate[TIME_LEN]; str collModify[TIME_LEN]; str collType[NAME_LEN]; struct *RescGrpInfo_PI; str collInfo1[MAX_NAME_LEN]; str collInfo2[MAX_NAME_LEN]; int *next;"
#define Rei_PI "int status; str statusStr[MAX_NAME_LEN]; str ruleName[NAME_LEN]; int *rsComm; struct *MsParamArray_PI; struct MsParamArray_PI; int l1descInx; struct *DataObjInp_PI; struct *DataObjInfo_PI; struct *RescGrpInfo_PI; struct *UserInfo_PI; struct *CollInfo_PI; struct *KeyValPair_PI; struct *UserInfo_PI; struct *CollInfo_PI; str ruleSet[RULE_SET_DEF_LENGTH]; int *next;"
#define ReArg_PI "int argc; str *myArgv[myArgc];"
#define ReiAndArg_PI "struct *Rei_PI; struct ReArg_PI;"
#define BytesBuf_PI "int buflen; char *buf(buflen);"
#define BinBytesBuf_PI "int buflen; bin *buf(buflen);"
#define MsParam_PI "str *label; piStr *type; %type *inOutStruct; struct *BinBytesBuf_PI;"
#define MsParamArray_PI "int paramLen; int oprType; struct *MsParam_PI[paramLen];"
#define TagStruct_PI "int ssLen; str *preTag[ssLen]; str *postTag[ssLen]; str *keyWord[ssLen];"
#define RodsObjStat_PI "double objSize; int objType; int dataMode; str dataId[NAME_LEN]; str chksum[NAME_LEN]; str ownerName[NAME_LEN]; str ownerZone[NAME_LEN]; str createTime[TIME_LEN]; str modifyTime[TIME_LEN]; struct SpecColl_PI;"
#define ReconnMsg_PI "int status; int cookie; int procState; int flag;"
```
#define VaultPathPolicy_PI "int scheme; int addUserName; int trimDirCnt;"
#define StrArray_PI "int len; int size; str *value(len)(size);"
#define IntArray_PI "int len; int *value(len);"
#define SpecColl_PI "int collClass; int type; str collection[MAX_NAME_LEN]; str
resource[MAX_NAME_LEN]; str cacheDir[MAX_NAME_LEN]; int cacheDirty; int replNum;"
#define SubFile_PI "struct RHostAddr_PI; str subFilePath[MAX_NAME_LEN]; int mode; int flags; double offset; struct *SpecColl_PI;"
#define XmsgTicketInfo_PI "int sendTicket; int rcvTicket; int expireTime; int flag;"
#define SendXmsgInfo_PI "int msgNumber; str msgType[HEADER_TYPE_LEN]; int numRcv; int flag; str *msg; int numDel; str *delAddress[numDel]; str *miscInfo;"
#define GetXmsgTicketInp_PI "int expireTime; int flag;"
#define SendXmsgInp_PI "struct XmsgTicketInfo_PI; str sendAddr[NAME_LEN]; struct SendXmsgInfo_PI;"
#define RcvXmsgInp_PI "int rcvTicket; int msgNumber; int seqNumber; str msgCondition[MAX_NAME_LEN];" /* XXXXX start of HDF5 PI */
#define h5error_PI "str major[MAX_ERROR_SIZE]; str minor[MAX_ERROR_SIZE];" /* XXXXX end of HDF5 PI */
#define h5File_PI "int fopID; str *filename; int ffid; struct *h5Group_PI; struct h5error_PI; int ftime;"
#define h5Group_PI "int gopID; int gfid; int nGroupMembers; struct *h5Group_PI(nGroupMembers); int nDatasetMembers; struct *h5Dataset_PI(nDatasetMembers); int nattributes; struct h5Attribute_PI(nattributes); struct h5error_PI; int gtime;"
#define h5Dataset_PI "int dopID; int dfid; int nvalue; int dtsize; int dclass; int nattributes; struct *dfullpath; struct *h5Attribute_PI(nattributes); struct h5Datatype_PI; struct h5Dataspace_PI; int dtime; float dvalue; char *value[nvalue]; double *value[nvalue]; struct h5error_PI;"
#define h5Attribute_PI "int aoPID; int afid; int aclass; struct *varName; struct *action; struct *var2CMap[nvalue]; double *varId[nvalue]; struct h5error_PI;"
#define h5Datatype_PI "int tclass; int tord; int tsign; int tsize; int ntmenbers; int *mtypes(ntmenbers); str *mnames[ntmenbers];" /* XXXXX need to fix the type dependence */
#define h5Dataspace_PI "int rank; int dims[H5S_MAX_RANK]; int npoints; int
start[H5DATASPACE_MAX_RANK]; int stride[H5DATASPACE_MAX_RANK]; int
count[H5DATASPACE_MAX_RANK];" /* content of collEnt_t cannot be freed since they are pointers in "value" */
#define CollEnt_PI "int objType; int replNum; int replStatus; int dataMode; double
dataSize; str $collName; str $dataName; str $dataId; str $createTime; str $modifyTime; str $chksum; str $resource; str $rescGrp; str $path; str $ownerName; struct SpecColl_PI;"
#define CollOprStat_PI "int filesCnt; int totalFileCnt; double bytesWritten; str
lastObjPath[MAX_NAME_LEN];" /* XXXXX need to fix the type dependence */
#define RuleStruct_PI "int maxNumOfRules; str *ruleBase[maxNumOfRules]; str *action[maxNumOfRules]; str *ruleHead[maxNumOfRules]; str *ruleCondition[maxNumOfRules]; str *ruleAction[maxNumOfRules]; str *ruleRecovery[maxNumOfRules]; double
ruleId[maxNumOfRules];" /* XXXXX need to fix the type dependence */
#define DNMapStruct_PI "int maxNumOfDVars; str *varName[maxNumOfDVars]; str
*funcVarName[maxNumOfDVars];" /* XXXXX need to fix the type dependence */
#define FNMapStruct_PI "int maxNumOfFMaps; str *funcName[maxNumOfFMaps]; str
*funcVarName[maxNumOfFMaps]; double *fmapId[maxNumOfFMaps];" /* XXXXX need to fix the type dependence */
#define MrvcStruct_PI "int maxNumOfMsrvcs; str *mrvcName[maxNumOfMsrvcs]; str
*mrvcVersion[maxNumOfMsrvcs]; str *mrvcHost[maxNumOfMsrvcs]; str *mrvcLocation[maxNumOfMsrvcs]; str
*mrvcLanguage[maxNumOfMsrvcs]; str *mrvcStatus[maxNumOfMsrvcs];" /* XXXXX need to fix the type dependence */
#define DataSeg_PI "double len; double offset;"
#define FileRestartInfo_PI "str fileName[MAX_NAME_LEN]; str objPath[MAX_NAME_LEN]; int
numSeg; int status; double fileSize; struct DataSeg_PI[MAX_NUM_CONFIG_TRAN_THR];" /* XXXXX need to fix the type dependence */
#define FileRestartInfo_PI "str fileName[MAX_NAME_LEN]; str objPath[MAX_NAME_LEN]; int
numSeg; int status; double fileSize; struct DataSeg_PI(numSeg);"
AUTHOR BIOGRAPHIES

The authors’ names are listed in alphabetical order [last, first, middle initial].

Moore, Reagan W.
Reagan Moore is a professor in the School of Information and Library Science at the University of North Carolina, Chapel Hill, chief scientist for Data Intensive Cyber Environments at the Renaissance Computing Institute, and director of the Data Intensive Cyber Environments Center at University of North Carolina. He coordinates research efforts in development of data grids, digital libraries, and preservation environments. Developed software systems include the Storage Resource Broker data grid and the integrated Rule-Oriented Data System. Supported projects include the National Archives and Records Administration Transcontinental Persistent Archive Prototype, and science data grids for seismology, oceanography, climate, high-energy physics, astronomy, and bioinformatics. An ongoing research interest is use of data grid technology to automate execution of management policies and validate trustworthiness of repositories. Dr. Moore’s previous roles include the following: director of the DICE group at the San Diego Supercomputer Center, and Manager of production services at SDSC. He previously worked as a computational plasma physicist at General Atomics on equilibrium and stability of toroidal fusion devices. He has a Ph.D. in plasma physics from the University of California, San Diego (1978), and a B.S. in physics from the California Institute of Technology (1967).

Rajasekar, Arcot
Arcot Rajasekar is a professor in the School of Library and Information Science at the University of North Carolina, Chapel Hill, and a chief scientist at the Renaissance Computing Institute (RENCI). Previously, he was at the San Diego Supercomputer Center at the University of California, San Diego, leading the Data Grids Technology Group. He has been involved in research and development of data grid middleware systems for over a decade and is a lead originator behind the concepts in the Storage Resource Broker (SRB) and the integrated Rule Oriented Data Systems (iRODS), two premier data grid middleware developed by the Data Intensive Cyber Environments Group. Dr. Rajasekar has a Ph.D. in computer science from the University of Maryland at College Park and has more than 100 publications in the areas of data grids, logic programming, deductive databases, digital library, and persistent archives.

Russell, Terrell
Terrell Russell recently graduated with his Ph.D. from the School of Information and Library Science at the University of North Carolina at Chapel Hill. He has been working with iRODS since 2008 and has been involved with writing microservices, testing, client interfaces, documentation, and project management. He is co-founder of claimID.com. His research focuses on identity, reputation, preservation, collective action, and expertise. He graduated from NC State with B.S. degrees in Computer Engineering and Information Technology and Service Organizations and an M.S. in Computer Networking.

Schroeder, Wayne
Wayne Schroeder leads the DICE (Data Intensive Cyber Environment) group of INC (Institute of Neural Science) at the University of California San Diego, is a senior software engineer with the iRODS team, and with Dr. Arcot Rajasekar and Michael Wan, designed and developed the core iRODS system. His iRODS contributions include the database catalog interface, administration tools, metadata interfaces, security/authentication, and installation and testing subsystems. He has over 35 years of experience in software engineering, with expertise in data management, computer security, networking, scientific applications, high performance computing, and system support/administration. Besides iRODS, career highlights include six years at LLNL (the Lawrence Livermore National Laboratory) - primarily at the NMF ECC (National Magnetic Fusion Energy Computer Center), many years at SDSC (the San Diego Supercomputer Center) where he contributed to its successful launch, managed the SDSC Central Systems Software group, and developed computer security software, and 2 years at a start-up (Entropia) helping to develop a cycle harvesting system. He earned a B.S. in computer science in 1976, magna cum laude, with a minor in psychology.
**de Torcy, Antoine**
Antoine de Torcy has been with the DICE group since 2003, first at the University of California, San Diego, and since 2008, at the University of North Carolina, Chapel Hill. Antoine has been involved in the development of Data Grid software, first with the Storage Resource Broker (SRB) and its successor, the Rule Oriented Data System (iRODS), implementing client interfaces for SRB and server modules for iRODS. He has been working on the Electronic Records Archive (ERA) project, using and extending iRODS to meet the list of ERA requirements, and building prototypes. His technical expertise has helped various groups build preservation environments based on iRODS and focused on data and metadata. Antoine holds an engineering degree in applied mathematics and computer science from the University of Paris–Dauphine.

**Wan, Michael**
Michael Wan leads the DICE (Data Intensive Cyber Environment) group of the INC (Institute of Neural Science) at the University of California, San Diego. He is the chief software architect of the Integrated Rule-Oriented Data System (iRODS) and the Storage Resource Broker (SRB). iRODS is a follow-on of SRB. It is a state-of-the-art production quality data grid software. The software supports data sharing, data publication, and data preservation. It is a rule-based system that can be used to automate the enforcement of data management policies and validation of assessment criteria. Before SRB, Michael has spent 10 years developing operating systems and archival storage systems at SDSC. Michael received his B.S. degree from Illinois State University and M.S. from Georgia Institute of Technology.

**Ward, Jewel H.**
Jewel Ward is a doctoral student at the University of North Carolina at Chapel Hill (UNC-CH), School of Information and Library Science. She is a Research Assistant for the Data Intensive Cyber Environments group. Since 2007, she has worked on projects ranging from interoperating a Social Science data archive with the integrated Rule-Oriented Data System (iRODS), to building a policy-based archivist's interface to iRODS. Prior to beginning the doctoral program, she was a Program Manager for the Digital Archive at the University of Southern California and a Post-master's Research Assistant in the Research Library at Los Alamos National Laboratory. Prior to obtaining her master's degree, she worked for several years in data communications, primarily as a Business Analyst, Web Developer and Technical Writer. She holds an M.S.I.S from the University of North Carolina at Chapel Hill, School of Information and Library Science, and an A.B. from the University of North Carolina at Chapel Hill in International Studies and History.

**Xu, Hao**
Hao Xu is a Ph.D. student at the Department of Computer Science, University of North Carolina at Chapel Hill. He is a research assistant in the Data Intensive Cyber Environments group. He has been working on improving the rule engine and the rule language of iRODS since 2010. His research interests include automatic theorem proving, programming languages, distributed data systems, and formal methods in software development. He graduated from Beihang University with a B.E. in Computer Science and Engineering and a B.S. minor in Applied Mathematics.
applyAllRules, 141
assign, 131
break, 132
CollByAdmin, 198
cut, 133
delay, 134
errorcode, 136
fail, 137
for, 139
foreach, 138
if, 140
listMS.r, 328
msiAclPolicy, 193
msiAddConditionToGenQuery, 194
msiAddKeyVal, 222
msiAddKeyValToMspStr, 104
msiAddSelectFieldToGenQuery, 195
msiAddUserToGroup, 196
msiAdmAddAppRuleStruct, 113
msiAdmAppendToTopOfCoreIRB, 114
msiAdmAppendToTopOfCoreRE, 115
msiAdmChangeCoreIRB, 116
msiAdmChangeCoreRE, 117
msiAdmClearAppRuleStruct, 118
msiAdmInsertRulesFromStructIntoDB, 119
msiAdmRetrieveRulesFromFileIntoStruct, 121
msiAdmRetrieveRulesFromDBIntoStruct, 127
msiAdmShowCoreRE, 122
msiAdmShowDVM, 123
msiAdmShowFNM, 124
msiAdmShowIRB, 125
msiAdmWriteRulesFromStructIntoFile, 126
msiApplyDCMetadataTemplate, 231
msiAssociateKeyValuePairsToObj, 223
msiAutoReplicateService, 61
msiCheckAccess, 81
msiCheckHostAccessControl, 162
msiCheckOwner, 82
msiCollCreate, 62
msiCollectionSpider, 251
msiCollRep, 63
msiCollRsync, 83
msiCommit, 197
msiConvertCurrency, 341
msiCopyAVUMetadata, 252
msiCreateUser, 199
msiCreateUserAccountsFromDataObj, 253
msiCreateXmsgInp, 151
msiDataObjAutoMove, 65
msiDataObjChksum, 84
msiDataObjChksumWithOptions, 85
msiDataObjClose, 75
msiDataObjCopy, 86
msiDataObjCreate, 76
msiDataObjGet, 87
msiDataObjGetWithOptions, 88
msiDataObjLseek, 77
msiDataObjOpen, 78
msiDataObjPhymv, 89
msiDataObjPut, 90
msiDataObjPutWithOptions, 92
msiDataObjRead, 79
msiDataObjRename, 93
msiDataObjRepl, 94
msiDataObjReplWithOptions, 96
msiDataObjRsync, 97
msiDataObjTrim, 98
msiDataObjUnlink, 99
msiDataObjWrite, 80
msiDboExec, 71
msiDbrCommit, 73
msiDbrRollback, 74
msiDeleteCollByAdmin, 200
msiDeleteDisallowed, 163
msiDeleteUnusedAVUs, 201
msiDeleteUser, 202
msiDeleteUsersFromDataObj, 254
msiDigestMonStat, 164
msiDoSomething, 105
msiExecCmd, 112
msiExecGenQuery, 203
msiExecStrCondQuery, 204
msiExecStrCondQueryWithOptions, 205
msiExit, 111
msiExportRecursiveCollMeta, 255
msiExtractNaraMetadata, 232
msiExtractTemplateMDFromBuf, 233
msiFlagDataObjwithAVU, 256
msiFlagInfectedObjs, 257
msiFlushMonStat, 166
msiFreeBuffer, 235
msiFtpGet, 339
msiGetAuditTrailInfoByActionID, 258
msiGetAuditTrailInfoByKeywords, 260
msiGetAuditTrailInfoByObjectID, 261
msiGetAuditTrailInfoByTime Stamp, 262
msiGetAuditTrailInfoByUserID, 263
msiGetCollectionACL, 264
msiGetCollectionContentsReport, 265
msiGetCollectionPSmeta, 266
msiGetCollectionSize, 267
msiGetContInxFromGenQueryOut, 206
msiGetDataObjACL, 269
msiGetDataObjAIP, 270
msiGetDataObjAVUs, 271
msiGetDataObjPSmeta, 272
msiGetDiffTime, 236
msiGetFormattedSystemTime, 289
msiStageDataObj, 190
msiStrArray2String, 227
msiStrchop, 129
msiString2KeyValPair, 228
msiString2StrArray, 229
msiStripAVUs, 287
msi_strlen, 128
msiStrToBytesBuf, 243
msiStructFileBundle, 288
msi_substr, 130
msiSysChksumDataObj, 191
msiSysMetaModify, 170
msiSysReplDataObj, 192
msiTarFileCreate, 69
msiTarFileExtract, 70
msiTwitterPost, 340
msiVacuum, 219
msi_WriteRodsLog, 110
msiXmlDocSchemaValidate, 347
msiXmsgCreateStream, 156
msiXmsgServerConnect, 157
msiXmsgServerDisConnect, 158
msiXsltApply, 350
msiz3950Submit, 352
nop, null, 144
print_hello, 145
readXMsg, 159
remote, 146
rulegenerateBagIt.r, 317
ruleintegrityACL.r, 320
ruleintegrityAVU.r, 322
ruleintegrityAVUvalue.r, 322
ruleintegrityExpiry.r, 325
ruleintegrityFileOwner.r, 327
ruleintegrityFileSize.r, 326
showCore.r, 329
succeed, 147
while, 148
writeBytesBuf, 244
writeKeyValPairs, 230
writeLine, 149
writePosInt, 245
writeString, 150
writeXMsg, 161