iRODS REST API User Documentation

Justin Kyle James, Raphaël Flores

Table of Contents

List of Tables

Chapter 1. Introduction

This is the official REST API for iRODS, and provides a simple, common API that can support diverse client use cases.

The iRODS REST API is designed to be:

- 'Proper' REST, with a resource orientation and sensible mapping of HTTP verbs to iRODS actions
- Compliant, with a JAX-RS compliant framework, and supporting content negotiation for requests and responses in both XML and JSON
- Flexible, by running in standard frameworks on commodity application servers, isolating the iRODS grid and running in a Java sandbox, requiring only an open 1247 port between the REST service layer and the iRODS data grid
- Well-tested, with reference tests for each function and use case that are continuously tested against each version of iRODS

This document describes the REST API, and the parameters and documents associated with each function.

1.1. Technology

The REST API is based on Java, and uses the Jargon API[1] to communicate with iRODS. The source code and project information for the REST API is available on GForge[2].

The REST API is built on top of the JBoss RESTEasy[3] framework, and deploys as a Java web application (.war). RESTEasy is a certified JAX-RS[4] implementation and as such is easily extended and customized using standard development practices.

1.2. Design Points, Resources and URL Paths

Like the Jargon API, the REST API is organized around the domain model of the iRODS catalog (ICAT). Consistent with REST, each of these iRODS catalog domains is roughly

[1] https://code.renci.org/gf/project/jargon/
[2] https://code.renci.org/gf/project/irods-rest/
[3] http://www.jboss.org/resteasy
[4] http://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services

equivalent to a resource. For example, an iRODS Collection, or directory, maps to the Collection object in Jargon, and is represented by a CollectionService in the REST API, responding to the /collection/ parent URL.

For Collections and Data Objects (data files), the iRODS logical path name is used to describe the iRODS file being acted upon. In this way, a REST operation to `GET` information on a collection has a path of:

http:/localhost/irods-rest/rest/collection/the/logical/irods/path

The collection in the URL indicates that this is an operation on a collection, and the remaining path information is used to resolve the iRODS logical path being acted upon.

Note that iRODS logical path names can be quite complex, and often contain special characters and spaces that are invalid in a URL. For this reason, the API supports optionally URL encoded iRODS absolute paths. This can be accomplished by URL encoding the logical path name before creating the URL string that will be sent to the REST service. The encoding is not necessary if there are no embedded spaces or special characters.

Continuing this resource orientation, the REST API follows the philosophy that each resource in the iRODS domain, such as Collection and Data Object, may have sub-resources that are particular to the parent resource at a given path. This means that AVU metadata services for a Collection in iRODS are a sub-resource of a given collection at a given path, and a URL such as the one below would be used to act on that sub-resource:

http:/localhost/irods-rest/rest/collection/the/logical/irods/path/metadata

In the above example, sending a `GET` request to that URL will return a listing of AVU metadata associated with a given iRODS collection.

When appropriate, a ticket string may be added to the URL parameter's ticket field for ticket based access to iRODS objects. The following is an example of using a ticket to get the file contents of a collection.

```
GET      http://user1:user1password@localhost:8080/irods-rest/rest/
collection/tempZone/home/rods?ticket=111222333444555
```

If the server is set up for anonymous access, the user is not required to provide credentials if requesting a resource via a ticket.

```
GET  http://localhost:8080/irods-rest/rest/collection/tempZone/home/
rods?ticket=11122233344455
```

1.2.1. HTTP Verbs and request parameters

Note that REST describes a set of HTTP verbs, and the iRODS REST API maps these verbs to resource as follows:

- `PUT` - creates a new resource at the given URI
- `POST` - updates a resource at the given URI, and often is used to create a sub-resource of the given URI
- `GET` - retrieves a resource or metadata about the resource
- `DELETE` - removes the resource at the given URI

Note that some arbitrary assignment of verbs is necessary due to technical or other considerations, but this will be documented and kept to a minimum.

The approach of this API is that HTTP parameters can be added to any requests in order to further refine the request or control the execution of the request or returned data. For example, a `GET` request to list children of a collection may include an offset parameter to page through multiple pages of children. The available parameters are documented along with each method.

Some requests, especially for sub-resources such as AVU metadata, are too complex to be placed into URLs or as parameters. For example, to add or delete AVU metadata associated with a collection, one uses a `POST` method and sends the required AVU actions as an XML or JSON request body. This approach for certain operations has the advantage of treating a set of metadata as a document, allowing the update of several attributes within one HTTP request. These operations that require a request body are documented with each method.

1.3. Authentication

The REST API for iRODS makes use of standard Basic Authentication[5] for HTTP requests. In this approach, the Authorization HTTP header is populated with username:password information that maps to the underlying iRODS account.

Basic Authentication passes these headers to the REST API with each request, and a new connection is made and authenticated as each request is made. Note that your deployed REST service should use an SSL certificate to secure this credential passing.

As noted in the ‘Installation and Configuration’ section, a deployment of the REST API is configured to point to a specific iRODS grid at deployment time, so the user and password

[5] http://en.wikipedia.org/wiki/Basic_access_authentication

information passed in via the headers are combined with the configured grid zone, host, port, and default storage resource information to derive the IRODSAccount used to authenticate to the data grid.

Currently, both STANDARD and PAM auth schemes are supported. The RestConfiguration object determines which scheme to use, and that option is then combined with the rest of the account configuration information to augment the user and password passed in through the basic auth processing.

In the following example, using the jargon-beans.xml file in src/main/resources to wire in custom configuration options, STANDARD authentication is indicated:

```
<beans:bean id="restConfiguration"
  class="org.irods.jargon.rest.configuration.RestConfiguration">
  <beans:property name="irodsHost" value="fedzone1.irods.org" />
  <beans:property name="irodsPort" value="1247" />
  <beans:property name="irodsZone" value="fedZone1" />
  <beans:property name="authType" value="STANDARD" /> <!--  STANDARD,PAM -->
  <beans:property name="defaultStorageResource" value="test1-resc" />
  <beans:property name="privateCertAbsPath" value="c:/temp/test-certs/private.pem" /
>
  <beans:property name="publicKeyAbsPath" value="c:/temp/test-certs/public.pem" />
  <beans:property name="webInterfaceURL"
   value="https://iren-web.renci.org:8443/idrop-web2" />
  <beans:property name="allowCors" value="false" />
  <beans:property name="corsAllowCredentials" value="false" />
  <beans:property name="corsOrigins">
   <util:list id="myList" value-type="java.lang.String">
    <beans:value>*</beans:value>
   </util:list>
  </beans:property>
  <beans:property name="corsMethods">
   <util:list id="myList" value-type="java.lang.String">
    <beans:value>GET</beans:value>
    <beans:value>POST</beans:value>
    <beans:value>DELETE</beans:value>
    <beans:value>PUT</beans:value>
   </util:list>
  </beans:property>

 </beans:bean>
```

Note that STANDARD and PAM auth may also be requested by prepending STANDARD* or PAM* to the user id portion of the Basic Authentication credentials. This will signal to the

REST service to opt for those specified auth methods, even if not configured in the spring xml configuration.

1.3.1. Content Negotiation for Requests

The REST API understands both XML and JSON for Request and Response bodies.

To receive JSON in response to a request, you may either:

- Set the Accept header in the request to ‘application/json;
- Set a parameter in the request URL of the form contentType="application/json"

To receive XML in the response, you may either:

- Set the Accept header in the request to ‘application/xml;
- Set a parameter in the request URL of the form contentType="application/xml"

To send XML in a request, set the Content-type header as follows:

"Content-Type", "application/xml"

To send JSON in a request, set the Content-type header as follows:

"Content-Type", "application/json"

Chapter 2. CORS Support

CORS is an acronym for Cross-Origin-Resource-Sharing. Overviews of CORS are out of the scope of this document, but W3C has a good reference here: http://www.w3.org/TR/cors/

The iRODS REST API has support for CORS, adding the appropriate headers when configured in the RestConfiguration class. The RestConfiguration class is wired in with Spring in the jargon-beans.xml file located in the src/main/resources folder of the REST source code.

The CORS configuration section of the jargon-beans.xml file looks like this:

```
<beans:property name="allowCors" value="true" />
  <beans:property name="corsAllowCredentials" value="false" />
  <beans:property name="corsOrigins">
   <util:list id="myList" value-type="java.lang.String">
    <beans:value>*</beans:value>
   </util:list>
  </beans:property>
  <beans:property name="corsMethods">
   <util:list id="myList" value-type="java.lang.String">
    <beans:value>GET</beans:value>
    <beans:value>POST</beans:value>
    <beans:value>DELETE</beans:value>
    <beans:value>PUT</beans:value>
   </util:list>
</beans:property>
```

Note that `allowCors` is a global setting, and if set to false no header processing is done. If set to 'true', then CORS headers for origin, methods, and allow credentials will be set in the response headers of the REST service.

If `allowCors` is set to true, and no corsOrigins are set, it will default to the origin of `'*'`. Otherwise, it will be set to the list of origins provided. Similarly, if corsMethods are not set, it will default to `GET`, `POST`, `DELETE`, `PUT`, otherwise, it will be set to the provided list.

Chapter 3. Installation and Configuration

The REST API is available as a .war file as indicated for each release in GitHub. This .war can be deployed on any standard servlet container, such as Jetty or Tomcat. Tomcat is the version we test with. The configuration of the REST API is necessary so it knows what host/port/zone/default resource, and default authentication method is in use. These presets are combined with the Basic Authentication credentials to connect to the target iRODS grid. This also prevents any installation of REST from being hijacked to talk to another iRODS grid!

The REST API .war file can be deployed to your servlet container,and once that is done, there is an /etc/irods-ext/irods-rest.properties file that must be put onto your server, readable by the servlet container. This .properties file looks like this:

```
irods.host=localhost
irods.port=1247
irods.zone=tempZone
utilize.packing.streams=true
auth.type=STANDARD
default.storage.resource=
web.interface.url=
cors.allow=true
```

Of course, the settings should correspond to your host. You will need to restart the servlet container or REST package to pick up these settings.

The REST API can be obtained via the GIT version control system, or via download, from the RENCI GForge[1] site. This package is built using Maven[2], and all necessary Jargon dependencies are available on the RENCI Maven repository.

When you download the irods-rest project, you can cd into that directory and issue the command:

```
>mvn package -Dmaven.test.skip=true
```

[1] https://code.renci.org/gf/project/irods-rest/
[2] http://maven.apache.org/

to build a deployable .war file that can then be installed on Tomcat, Jetty, Glassfish, or any other compliant servlet container. Note the flag to skip the unit test phase, which requires further configuration to run.

The iRODS REST API uses Spring for wiring and configuration, and before packaging your application, you should edit the jargon-beans.xml file under src/main/resources to point to your target grid. For example, to run the REST API against an iRODS grid on fedZone1, the configuration would look like this:

```
<beans:bean id="restConfiguration"
  class="org.irods.jargon.rest.configuration.RestConfiguration">
  <beans:property name="irodsHost" value="fedZone1" />
  <beans:property name="irodsPort" value="1247" />
  <beans:property name="irodsZone" value="fedZone1" />
  <beans:property name="defaultStorageResource" value="test1-resc" />
  <beans:property name="webInterfaceURL"
   value="https://iren-web.renci.org:8443/idrop-web2" />
</beans:bean>
```

Note the standard iRODS grid configuration as well as an optional URL to an installation of iDROP Web. (Work in Progress here).

Note that as this API develops, we'll probably add an ability today? configure the REST API via an /etc/idrop-web configuration file like the web interface.

Once the API is configured, and then built with Maven, it may be deployed, and then accessed at the URL you configure. The typical URL is something like:

http://host:port/irods-rest/rest/resource/extrapathinformation

A handy way to test via a browser is to use the 'ping' service by going to a browser and entering a request like:

http://localhost:8081/irods-rest-0.0.1-SNAPSHOT/rest/server

Which is a `GET` service that will ping the server and return some basic server information. For example, in XML, it should return something similar to this:

```
<ns2:serverInfo xmlns:ns2="http://irods.org/irods-rest">
<apiVersion>d</apiVersion>
<currentServerTime>1388922589000</currentServerTime>
```

```
<icatEnabled>ICAT_ENABLED</icatEnabled>
<initializeDate>2014-01-06T10:32:54.722-05:00</initializeDate>
<relVersion>rods3.3</relVersion>
<rodsZone>fedZone1</rodsZone>
<serverBootTime>1387383118</serverBootTime>
</ns2:serverInfo>
```

This action should require you to provide the iRODS user name and password in a Basic Authentication dialog before retrieving that information. This verifies that the service is working!

Chapter 4. Operations on Collections

4.1. Get Collection Information

4.1.1. Description

This `GET` operation will retrieve basic catalog metadata about the iRODS collection. This method can also optionally provide a pageable listing of the child collections and data objects underneath the given collection.

4.1.2. Requests

Syntax

`GET /collection/irodsabsolutepathtocollection Request Parameters`

Table 4.1. Collection information

Name	Description
offset	`number >=0` that indicates the offset into child collections or data objects when displaying children
listing	`true` if a listing of children of this collection should be provided in the response
listType	`both|collections|data` that indicates the type of listing. Using `both` will list all data objects and collections. Subsequent pages of data objects or collections are retrieved by sending `collections` or `data` with an offset. Default = `both`

Responses - XML

```
<ns2:collection xmlns:ns2="http://irods.org/irods-rest" collectionId="187864">
<children count="1" id="187865" lastResult="true" specColType="NORMAL"
 totalRecords="5">
```

```
<createdAt>2013-08-09T23:10:41-04:00</createdAt>
<dataSize>345217</dataSize>
<modifiedAt>2013-08-09T23:10:41-04:00</modifiedAt>
<objectType>DATA_OBJECT</objectType>
<ownerName>rods</ownerName>
<ownerZone>fedZone1</ownerZone>
<parentPath>/fedZone1/home/rods/shared</parentPath>
<pathOrName>cpylog</pathOrName>
<specialObjectPath/>
</children>
...
<collectionInheritance>1</collectionInheritance>
<collectionMapId>0</collectionMapId>
<collectionName>/fedZone1/home/rods/shared</collectionName>
<collectionOwnerName>rods</collectionOwnerName>
<collectionOwnerZone>fedZone1</collectionOwnerZone>
<collectionParentName>/fedZone1/home/rods/</collectionParentName>
<comments/>
<createdAt>2013-08-09T23:10:15-04:00</createdAt>
<info1/>
<info2/>
<modifiedAt>2013-08-28T21:50:06-04:00</modifiedAt>
<objectPath/>
<specColType>NORMAL</specColType>
</ns2:collection>
```

Responses - JSON

```
{
   "collectionId":187864,
   "collectionName":"/fedZone1/home/rods/shared",
   "objectPath":"",
   "collectionParentName":"/fedZone1/home/rods/",
   "collectionOwnerName":"rods",
   "collectionOwnerZone":"fedZone1",
   "collectionMapId":"0",
   "collectionInheritance":"1",
   "comments":"",
   "info1":"",
   "info2":"",
   "createdAt":1376104215000,
   "modifiedAt":1377741006000,
   "specColType":"NORMAL",
   "children":[
      {
         "parentPath":"/fedZone1/home/rods/shared",
         "pathOrName":"cpylog",
```

```
        "specialObjectPath":"",
        "objectType":"DATA_OBJECT",
        "createdAt":1376104241000,
        "modifiedAt":1376104241000,
        "dataSize":345217,
        "ownerName":"rods",
        "ownerZone":"fedZone1",
        "id":187865,
        "specColType":"NORMAL",
        "count":1,
        "lastResult":true,
        "totalRecords":5
     }
  ]
}
```

4.2. Create a New Collection

4.2.1. Description

This `PUT` operation will first create a new collection, and then return back a description of the iRODS catalog entry for the newly created collection. This is an idempotent method, and if the collection already exists, the data for the existing collection is returned.

4.2.2. Requests

Syntax

`PUT /collection/irodsabsolutepathtocollection Request Parameters`

Table 4.2. Collection creation

Name	Description
n/a	

Responses - XML

```
<ns2:collection xmlns:ns2="http://irods.org/irods-rest" collectionId="187864">
<collectionInheritance>1</collectionInheritance>
<collectionMapId>0</collectionMapId>
<collectionName>/fedZone1/home/rods/shared</collectionName>
<collectionOwnerName>rods</collectionOwnerName>
<collectionOwnerZone>fedZone1</collectionOwnerZone>
```

```
<collectionParentName>/fedZone1/home/rods/</collectionParentName>
<comments/>
<createdAt>2013-08-09T23:10:15-04:00</createdAt>
<info1/>
<info2/>
<modifiedAt>2013-08-28T21:50:06-04:00</modifiedAt>
<objectPath/>
<specColType>NORMAL</specColType>
</ns2:collection>
```

Responses - JSON

```
{
   "collectionId":187864,
   "collectionName":"/fedZone1/home/rods/shared",
   "objectPath":"",
   "collectionParentName":"/fedZone1/home/rods/",
   "collectionOwnerName":"rods",
   "collectionOwnerZone":"fedZone1",
   "collectionMapId":"0",
   "collectionInheritance":"1",
   "comments":"",
   "info1":"",
   "info2":"",
   "createdAt":1376104215000,
   "modifiedAt":1377741006000,
   "specColType":"NORMAL",
 }
```

4.3. Delete a Collection

4.3.1. Description

This `DELETE` operation will remove the given collection. A `force` option is provided with an additional request parameter. This method will silently ignore deletes of non-existent collections.

Note that `DELETE` requests do not return a body in HTTP. This method will instead return a `204 No Content` response.

4.3.2. Requests

Syntax

```
DELETE /collection/irodsabsolutepathtocollection
```

Request Parameters

Table 4.3. Collection deletion

Name	Description
force	`true` to use a force option Default = `false`

Responses - XML

n/a

Responses - JSON

n/a

Chapter 5. Operations on Collection AVU Metadata

The following operations apply to the AVU sub-resource of an iRODS collection, and represents AVU metadata attached to the given collection.

5.1. Get Collection AVU Metadata

5.1.1. Description

This `GET` operation will retrieve the AVU metadata associated with an iRODS parent collection

5.1.2. Requests

Syntax

```
GET /collection/irodsabsolutepathtocollection/metadata
```

Request Parameters

Table 5.1. Collection AVU Metadata

Name	Description
n/a	

Responses - XML

```
<ns2:metadataListing xmlns:ns2="http://irods.org/irods-rest"
 objectType="COLLECTION">
<metadataEntries count="1" lastResult="true" totalRecords="0">
<attribute>attr1</attribute>
<unit>unit1</unit>
<value>val1</value>
</metadataEntries>
<uniqueNameString>fedZone1/home/rods/shared</uniqueNameString>
</ns2:metadataListing>
```

Responses - JSON

```
{
   "metadataEntries":[
      {
         "attribute":"attr1",
         "value":"val1",
         "unit":"unit1",
         "count":1,
         "lastResult":true,
         "totalRecords":0
      }
   ],
   "objectType":"COLLECTION",
   "uniqueNameString":"fedZone1/home/rods/shared"
}
```

5.2. Add Collection AVU Metadata

5.2.1. Description

This `PUT` operation will bulk add the provided AVU metadata associated with an iRODS parent collection.

This bulk operation requires a request body in XML or JSON, as AVU metadata is too large and complex for proper expression as a URL or parameter. This bulk mode also is more efficient for larger amounts of metadata, requiring fewer round trips.

Note that the response will detail the disposition, and any errors that occurred for individual AVU values.

5.2.2. Requests

Syntax

`PUT /collection/irodsabsolutepathtocollection/metadata`

Request Parameters

Table 5.2. Collection AVU Metadata addition

Name	Description
n/a	

Request Body - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:metadataOperation xmlns:ns2="http://irods.org/irods-rest">
 <metadataEntries>
  <attribute>testBulkAddCollectionAVUSendXMLAttr1</attribute>
  <unit>testBulkAddCollectionAVUSendXMLUnit1</unit>
  <value>testBulkAddCollectionAVUSendXMLValue1</value>
 </metadataEntries>
 <metadataEntries>
  <attribute>testBulkAddCollectionAVUSendXMLAttr2</attribute>
  <unit>testBulkAddCollectionAVUSendXMLUnit2</unit>
  <value>testBulkAddCollectionAVUSendXMLValue2</value>
 </metadataEntries>
</ns2:metadataOperation>
```

Request Body - JSON

```
{
  "metadataEntries": [
    {
      "attribute": "testBulkAddCollectionAVUJsonAttr1",
      "value": "testBulkAddCollectionAVUJsonValue1",
      "unit": "testBulkAddCollectionAVUJsonUnit1"
    },
    {
      "attribute": "testBulkAddCollectionAVUJsonAttr2",
      "value": "testBulkAddCollectionAVUJsonValue2",
      "unit": "testBulkAddCollectionAVUJsonUnit2"
    }
  ]
}
```

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<collection xmlns:ns2="http://irods.org/irods-rest">
 <ns2:metadataOperationResultEntry
  resultStatus="OK">
  <attributeString>testBulkAddCollectionAVUSendXMLAttr1</attributeString>
  <message />
  <unit>testBulkAddCollectionAVUSendXMLUnit1</unit>
  <valueString>testBulkAddCollectionAVUSendXMLValue1</valueString>
 </ns2:metadataOperationResultEntry>
 <ns2:metadataOperationResultEntry
  resultStatus="OK">
  <attributeString>testBulkAddCollectionAVUSendXMLAttr2</attributeString>
```

```
  <message />
  <unit>testBulkAddCollectionAVUSendXMLUnit2</unit>
  <valueString>testBulkAddCollectionAVUSendXMLValue2</valueString>
 </ns2:metadataOperationResultEntry>
</collection>
```

Responses - JSON

```
[
  {
    "attributeString": "testBulkAddCollectionAVUJsonAttr1",
    "valueString": "testBulkAddCollectionAVUJsonValue1",
    "unit": "testBulkAddCollectionAVUJsonUnit1",
    "resultStatus": "OK",
    "message": ""
  },
  {
    "attributeString": "testBulkAddCollectionAVUJsonAttr2",
    "valueString": "testBulkAddCollectionAVUJsonValue2",
    "unit": "testBulkAddCollectionAVUJsonUnit2",
    "resultStatus": "OK",
    "message": ""
  }
]
```

5.3. Delete Collection AVU Metadata

5.3.1. Description

This `POST` operation will bulk delete the provided AVU metadata associated with an iRODS parent collection. `POST` is used here as an HTTP `DELETE` action cannot have a body.

This bulk operation requires a request body in XML or JSON, as AVU metadata is too large and complex for proper expression as a URL or parameter. This bulk mode also is more efficient for larger amounts of metadata, requiring fewer round trips.

Note that the response will detail the disposition, and any errors that occurred for individual AVU values.

5.3.2. Requests

Syntax

`POST /collection/irodsabsolutepathtocollection/metadata`

Request Parameters

Table 5.3. Collection AVU Metadata deletion

Name	Description
n/a	

Request Body - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:metadataOperation xmlns:ns2="http://irods.org/irods-rest">
 <metadataEntries>
  <attribute>testBulkAddCollectionAVUSendXMLAttr1</attribute>
  <unit>testBulkAddCollectionAVUSendXMLUnit1</unit>
  <value>testBulkAddCollectionAVUSendXMLValue1</value>
 </metadataEntries>
 <metadataEntries>
  <attribute>testBulkAddCollectionAVUSendXMLAttr2</attribute>
  <unit>testBulkAddCollectionAVUSendXMLUnit2</unit>
  <value>testBulkAddCollectionAVUSendXMLValue2</value>
 </metadataEntries>
</ns2:metadataOperation>
```

Request Body - JSON

```
{
  "metadataEntries": [
    {
      "attribute": "testBulkAddCollectionAVUJsonAttr1",
      "value": "testBulkAddCollectionAVUJsonValue1",
      "unit": "testBulkAddCollectionAVUJsonUnit1"
    },
    {
      "attribute": "testBulkAddCollectionAVUJsonAttr2",
      "value": "testBulkAddCollectionAVUJsonValue2",
      "unit": "testBulkAddCollectionAVUJsonUnit2"
    }
  ]
}
```

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<collection xmlns:ns2="http://irods.org/irods-rest">
```

```
 <ns2:metadataOperationResultEntry
  resultStatus="OK">
  <attributeString>testBulkAddCollectionAVUSendXMLAttr1</attributeString>
  <message />
  <unit>testBulkAddCollectionAVUSendXMLUnit1</unit>
  <valueString>testBulkAddCollectionAVUSendXMLValue1</valueString>
 </ns2:metadataOperationResultEntry>
 <ns2:metadataOperationResultEntry
  resultStatus="OK">
  <attributeString>testBulkAddCollectionAVUSendXMLAttr2</attributeString>
  <message />
  <unit>testBulkAddCollectionAVUSendXMLUnit2</unit>
  <valueString>testBulkAddCollectionAVUSendXMLValue2</valueString>
 </ns2:metadataOperationResultEntry>
</collection>
```

Responses - JSON

```
[
  {
    "attributeString": "testBulkAddCollectionAVUJsonAttr1",
    "valueString": "testBulkAddCollectionAVUJsonValue1",
    "unit": "testBulkAddCollectionAVUJsonUnit1",
    "resultStatus": "OK",
    "message": ""
  },
  {
    "attributeString": "testBulkAddCollectionAVUJsonAttr2",
    "valueString": "testBulkAddCollectionAVUJsonValue2",
    "unit": "testBulkAddCollectionAVUJsonUnit2",
    "resultStatus": "OK",
    "message": ""
  }
]
```

Chapter 6. Operations on Collection ACLs (Permissions)

The following operations apply to the ACL sub-resource of iRODS collections, and can alter access permissions.

6.1. Get Collection Permissions

6.1.1. Description

This `GET` operation will retrieve the permissions associated with an iRODS collection

6.1.2. Requests

Syntax

```
GET /collection/irodsabsolutepathtocollection/acl
```

Request Parameters

Table 6.1. Collection ACL

Name	Description
n/a	

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:permissionListing xmlns:ns2="http://irods.org/irods-rest">
 <absolutePathString>/fedZone1/home/test1/jargon-scratch/RestCollectionServiceTest/
testGetCollectionAclXML
 </absolutePathString>
 <inheritance>true</inheritance>
 <objectType>COLLECTION</objectType>
 <permissionEntries>
  <filePermissionEnum>OWN</filePermissionEnum>
  <userId>10007</userId>
  <userName>rods</userName>
  <userType>RODS_ADMIN</userType>
  <userZone>fedZone1</userZone>
 </permissionEntries>
```

```
 <permissionEntries>
  <filePermissionEnum>OWN</filePermissionEnum>
  <userId>10012</userId>
  <userName>test1</userName>
  <userType>RODS_ADMIN</userType>
  <userZone>fedZone1</userZone>
 </permissionEntries>
</ns2:permissionListing>
```

Responses - JSON

```
{
  "permissionEntries": [
    {
      "userName": "rods",
      "userZone": "fedZone1",
      "userId": "10007",
      "userType": "RODS_ADMIN",
      "filePermissionEnum": "OWN"
    },
    {
      "userName": "test1",
      "userZone": "fedZone1",
      "userId": "10012",
      "userType": "RODS_ADMIN",
      "filePermissionEnum": "OWN"
    }
  ],
  "objectType": "COLLECTION",
  "absolutePathString": "/fedZone1/home/test1/jargon-scratch/
RestCollectionServiceTest/testGetCollectionAclJson",
  "inheritance": true
}
```

6.2. Add Collection Permission

6.2.1. Description

This `PUT` operation will set a collection permission. Note that this is an idempotent method, and if an existing permission is already stored, the new permission will replace it. This means updating a permission is accomplished using this same method.

Note that this method returns no body, and a normal operation returns an `HTTP 204 response code`.

6.2.2. Requests

Syntax

```
PUT /collection/irodsabsolutepathtocollection/acl/username
```

Note on username: iRODS supports a `user#zone` format to describe user names. This special format is useful when defining permissions on a federated grid. In normal circumstances, just the user name is required. If the zone information is also required, it should be expressed in the username portion of the url path in `username,zone` format rather than `username#zone` format. This helps clarify the URL and prevents conflicts with the URL anchor pattern.

Request Parameters

Table 6.2. Collection ACL addition

Name	Description
recursive	Indicates whether the operation should be applied recursively. (`true` | `false`) The default is `false`
permission	The permission value to set (`READ` | `WRITE` | `OWN`). The default is READ

Responses - XML

n/a

Responses - JSON

n/a

6.3. Remove Collection Permission

6.3.1. Description

This `DELETE` operation will remove a collection permission. Note that this is an idempotent method, and if no permission exists, it will silently ignore the request and return a normal response.

This method returns no body, and a normal operation returns an `HTTP 204 response code`.

6.3.2. Requests

Syntax

```
DELETE /collection/irodsabsolutepathtocollection/acl/username
```

note on username: iRODS supports a `user#zone` format to describe user names. This special format is useful when defining permissions on a federated grid. In normal circumstances, just the user name is required. If the zone information is also required, it should be expressed in the username portion of the url path in `username,zone` format rather than `username#zone` format. This helps clarify the URL and prevents conflicts with the URL anchor pattern._

Request Parameters

Table 6.3. Collection ACL deletion

Name	Description
recursive	Indicates whether the operation should be applied recursively. (`true` | `false`) The default is `false`

Responses - XML

n/a

Responses - JSON

n/a

Chapter 7. Operations on Data Objects

The following operations concern iRODS Data Objects, which are files, as opposed to directories).

7.1. Get Data Object Information

7.1.1. Description

This `GET` operation will retrieve basic catalog metadata about the iRODS Data Object. ==== Requests

Syntax

```
GET /dataObject/irodsabsolutepathtodataobject.extension
```

Request Parameters

Name	Description
n/a	

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:dataObject xmlns:ns2="http://irods.org/irods-rest"
 collectionId="614201" id="614202">
 <checksum />
 <collectionName>/fedZone1/home/test1/jargon-scratch/RestDataObjectServiceTest
 </collectionName>
 <comments />
 <createdAt>2014-01-06T03:58:35-05:00</createdAt>
 <dataMapId>0</dataMapId>
 <dataName>testGetDataObjectDataXML.dat</dataName>
 <dataOwnerName>test1</dataOwnerName>
 <dataOwnerZone>fedZone1</dataOwnerZone>
 <dataPath>/opt/iRODS/iRODS3.3/Vault1/home/test1/jargon-scratch/
RestDataObjectServiceTest/testGetDataObjectDataXML.dat
 </dataPath>
 <dataReplicationNumber>0</dataReplicationNumber>
 <dataSize>1</dataSize>
 <dataStatus />
 <dataTypeName>generic</dataTypeName>
```

```
 <dataVersion>0</dataVersion>
 <expiry />
 <objectPath />
 <replicationStatus>1</replicationStatus>
 <resourceGroupName />
 <resourceName>test1-resc</resourceName>
 <specColType>NORMAL</specColType>
 <updatedAt>2014-01-06T03:58:35-05:00</updatedAt>
</ns2:dataObject>
```

Responses - JSON

```
{
  "id": 614206,
  "collectionId": 614205,
  "dataName": "testFindByAbsolutePath.dat",
  "collectionName": "/fedZone1/home/test1/jargon-scratch/RestDataObjectServiceTest",
  "dataReplicationNumber": 0,
  "dataVersion": 0,
  "dataTypeName": "generic",
  "dataSize": 0,
  "resourceGroupName": "",
  "resourceName": "test1-resc",
  "dataPath": "/opt/iRODS/iRODS3.3/Vault1/home/test1/jargon-scratch/
RestDataObjectServiceTest/testFindByAbsolutePath.dat",
  "dataOwnerName": "test1",
  "dataOwnerZone": "fedZone1",
  "replicationStatus": "1",
  "dataStatus": "",
  "checksum": "",
  "expiry": "",
  "dataMapId": 0,
  "comments": "",
  "createdAt": 1388998844000,
  "updatedAt": 1388998844000,
  "specColType": "NORMAL",
  "objectPath": ""
}
```

7.1.2. Requests

Syntax

`DELETE /dataObject/irodsabsolutepathtodataobject.extension`

Request Parameters

Name	Description
n/a	

Responses - XML

n/a

Responses - JSON

n/a

7.2. Get Data Object File Content (File Download)

7.2.1. Description

This `GET` operation will retrieve the actual contents of a Data Object in iRODS. This causes an HTTP file download action. Note that the service uses a parent resource of 'fileContents' as opposed to 'dataObject'. This is to preserve the symmetry between upload (POST) and download (GET) while allowing `GET` for a Data Object to return the catalog metadata instead of the contents.

Should we here consider the file contents as a sub-resource instead? This might make it more consistent?

The response to this request will be an application/octet-stream with the binary file data. Here is an example snippet in Java, from the JUnit tests in the FileContentsServiceTest using the Apache HTTPClient library:

```
HttpGet httpGet = new HttpGet(sb.toString());

HttpResponse response = clientAndContext.getHttpClient().execute(
    httpGet, clientAndContext.getHttpContext());

HttpEntity entity = response.getEntity();
long len = 0;
InputStream inputStream = null;

if (entity != null) {
 len = entity.getContentLength();
 inputStream = entity.getContent();
 // write the file to wherever you want it.
```

```
}
```

7.2.2. Requests

Syntax

`GET /fileContents/irodsabsolutepathtodataobject.extension`

Request Parameters

TODO: add param to download segments of a file with offset and length

Table 7.1. Data object file content retrieval

Name	Description
n/a	

Responses - XML

n/a

Responses - JSON

n/a

7.3. Post Data Object File Content (File Upload)

7.3.1. Description

This `POST` operation will upload binary data to the actual contents of a Data Object in iRODS. This causes an HTTP multipart upload action.

Should we here consider the file contents as a sub-resource instead? This might make it more consistent?

The request should be a `POST` of multipart form data, with the attached file set to the form parameter uploadFile. The following JUnit test snippet in the FileContentsServiceTest, illustrates an upload using the Apache HTTPClient library.

```
HttpPost httpPost = new HttpPost(sb.toString());
httpPost.addHeader("accept", "application/json");
// httpPost.addHeader("Content-type", "multipart/form-data");
FileBody fileEntity = new FileBody(localFile,
    "application/octet-stream");
MultipartEntity reqEntity = new MultipartEntity(
    HttpMultipartMode.BROWSER_COMPATIBLE);
reqEntity.addPart("uploadFile", fileEntity);
httpPost.setEntity(reqEntity);
HttpResponse response = clientAndContext.getHttpClient().execute(
httpPost, clientAndContext.getHttpContext());
```

Note that the response to this operation is equivalent to the `GET` of the Data Object catalog metadata, reflecting the file that was uploaded to iRODS.

7.3.2. Requests

Syntax

`POST /fileContents/irodsabsolutepathtodataobject.extension`

Request Parameters

TODO: add param to upload segments of a file with offset and length

Name	Description
n/a	

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:dataObject xmlns:ns2="http://irods.org/irods-rest"
 collectionId="614201" id="614202">
 <checksum />
 <collectionName>/fedZone1/home/test1/jargon-scratch/RestDataObjectServiceTest
 </collectionName>
 <comments />
 <createdAt>2014-01-06T03:58:35-05:00</createdAt>
 <dataMapId>0</dataMapId>
 <dataName>testGetDataObjectDataXML.dat</dataName>
 <dataOwnerName>test1</dataOwnerName>
```

```
 <dataOwnerZone>fedZone1</dataOwnerZone>
 <dataPath>/opt/iRODS/iRODS3.3/Vault1/home/test1/jargon-scratch/
RestDataObjectServiceTest/testGetDataObjectDataXML.dat
 </dataPath>
 <dataReplicationNumber>0</dataReplicationNumber>
 <dataSize>1</dataSize>
 <dataStatus />
 <dataTypeName>generic</dataTypeName>
 <dataVersion>0</dataVersion>
 <expiry />
 <objectPath />
 <replicationStatus>1</replicationStatus>
 <resourceGroupName />
 <resourceName>test1-resc</resourceName>
 <specColType>NORMAL</specColType>
 <updatedAt>2014-01-06T03:58:35-05:00</updatedAt>
</ns2:dataObject>
```

Responses - JSON

```
{
  "id": 614206,
  "collectionId": 614205,
  "dataName": "testFindByAbsolutePath.dat",
  "collectionName": "/fedZone1/home/test1/jargon-scratch/RestDataObjectServiceTest",
  "dataReplicationNumber": 0,
  "dataVersion": 0,
  "dataTypeName": "generic",
  "dataSize": 1,
  "resourceGroupName": "",
  "resourceName": "test1-resc",
  "dataPath": "/opt/iRODS/iRODS3.3/Vault1/home/test1/jargon-scratch/
RestDataObjectServiceTest/testFindByAbsolutePath.dat",
  "dataOwnerName": "test1",
  "dataOwnerZone": "fedZone1",
  "replicationStatus": "1",
  "dataStatus": "",
  "checksum": "",
  "expiry": "",
  "dataMapId": 0,
  "comments": "",
  "createdAt": 1388998844000,
  "updatedAt": 1388998844000,
  "specColType": "NORMAL",
  "objectPath": ""
}
```

Chapter 8. Operations on Data Object AVU Metadata

The following operations apply to the AVU sub-resource of an iRODS data objects, and represents AVU metadata attached to the given data object

8.1. Get Data Object AVU Metadata

8.1.1. Description

This `GET` operation will retrieve the AVU metadata associated with an iRODS data object

8.1.2. Requests

Syntax

```
GET /dataObject/irodsabsolutepathtodataobject.extension/metadata
```

Request Parameters

Name	Description
n/a	

Responses - XML

```
<ns2:metadataListing xmlns:ns2="http://irods.org/irods-rest"
 objectType="DATA_OBJECT">
<metadataEntries count="1" lastResult="true" totalRecords="0">
<attribute>attr1</attribute>
<unit>unit1</unit>
<value>val1</value>
</metadataEntries>
<uniqueNameString>fedZone1/home/rods/shared</uniqueNameString>
</ns2:metadataListing>
```

Responses - JSON

```
{
   "metadataEntries":[
```

```
      {
         "attribute":"attr1",
         "value":"val1",
         "unit":"unit1",
         "count":1,
         "lastResult":true,
         "totalRecords":0
      }
   ],
   "objectType":'DATA_OBJECT",
   "uniqueNameString":"fedZone1/home/rods/shared"
}
```

8.2. Add Data Object AVU Metadata

8.2.1. Description

This `PUT` operation will bulk add the provided AVU metadata associated with an iRODS data object.

This bulk operation requires a request body in XML or JSON, as AVU metadata is too large and complex for proper expression as a URL or parameter. This bulk mode also is more efficient for larger amounts of metadata, requiring fewer round trips.

Note that the response will detail the disposition, and any errors that occurred for individual AVU values.

8.2.2. Requests

Syntax

`PUT /dataObject/irodsabsolutepathtodataobject.extension/metadata`

Request Parameters

Name	Description
n/a	

Request Body - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:metadataOperation xmlns:ns2="http://irods.org/irods-rest">
 <metadataEntries>
```

```
  <attribute>testBulkAddCollectionAVUSendXMLAttr1</attribute>
  <unit>testBulkAddCollectionAVUSendXMLUnit1</unit>
  <value>testBulkAddCollectionAVUSendXMLValue1</value>
 </metadataEntries>
 <metadataEntries>
  <attribute>testBulkAddCollectionAVUSendXMLAttr2</attribute>
  <unit>testBulkAddCollectionAVUSendXMLUnit2</unit>
  <value>testBulkAddCollectionAVUSendXMLValue2</value>
 </metadataEntries>
</ns2:metadataOperation>
```

Request Body - JSON

```
{"metadataEntries":
[{"attribute":"testBulkAddCollectionAVUJsonAttr1","value":"testBulkAddCollectionAVUJsonValue1
{"attribute":"testBulkAddCollectionAVUJsonAttr2","value":"testBulkAddCollectionAVUJsonValue2"
```

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<collection xmlns:ns2="http://irods.org/irods-rest">
 <ns2:metadataOperationResultEntry
  resultStatus="OK">
  <attributeString>testBulkAddCollectionAVUSendXMLAttr1</attributeString>
  <message />
  <unit>testBulkAddCollectionAVUSendXMLUnit1</unit>
  <valueString>testBulkAddCollectionAVUSendXMLValue1</valueString>
 </ns2:metadataOperationResultEntry>
 <ns2:metadataOperationResultEntry
  resultStatus="OK">
  <attributeString>testBulkAddCollectionAVUSendXMLAttr2</attributeString>
  <message />
  <unit>testBulkAddCollectionAVUSendXMLUnit2</unit>
  <valueString>testBulkAddCollectionAVUSendXMLValue2</valueString>
 </ns2:metadataOperationResultEntry>
</collection>
```

Responses - JSON

```
[
  {
    "attributeString": "testBulkAddCollectionAVUJsonAttr1",
    "valueString": "testBulkAddCollectionAVUJsonValue1",
    "unit": "testBulkAddCollectionAVUJsonUnit1",
    "resultStatus": "OK",
```

```
    "message": ""
  },
  {
    "attributeString": "testBulkAddCollectionAVUJsonAttr2",
    "valueString": "testBulkAddCollectionAVUJsonValue2",
    "unit": "testBulkAddCollectionAVUJsonUnit2",
    "resultStatus": "OK",
    "message": ""
  }
]
```

8.3. Delete Data Object AVU Metadata

8.3.1. Description

This `POST` operation will bulk delete the provided AVU metadata associated with an iRODS data object. `POST` is used here as an HTTP `DELETE` action cannot have a body.

This bulk operation requires a request body in XML or JSON, as AVU metadata is too large and complex for proper expression as a URL or parameter. This bulk mode also is more efficient for larger amounts of metadata, requiring fewer round trips.

Note that the response will detail the disposition, and any errors that occurred for individual AVU values.

8.3.2. Requests

Syntax

`POST /dataObject/irodsabsolutepathtodataobject.extension/metadata`

Request Parameters

Name	Description
n/a	

Request Body - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:metadataOperation xmlns:ns2="http://irods.org/irods-rest">
 <metadataEntries>
  <attribute>testBulkAddCollectionAVUSendXMLAttr1</attribute>
  <unit>testBulkAddCollectionAVUSendXMLUnit1</unit>
```

```
  <value>testBulkAddCollectionAVUSendXMLValue1</value>
 </metadataEntries>
 <metadataEntries>
  <attribute>testBulkAddCollectionAVUSendXMLAttr2</attribute>
  <unit>testBulkAddCollectionAVUSendXMLUnit2</unit>
  <value>testBulkAddCollectionAVUSendXMLValue2</value>
 </metadataEntries>
</ns2:metadataOperation>
```

Request Body - JSON

```
{
  "metadataEntries": [
    {
      "attribute": "attr1",
      "value": "val1",
      "unit": "unit1",
      "count": 1,
      "lastResult": true,
      "totalRecords": 0
    }
  ],
  "objectType": "DATA_OBJECT",
  "uniqueNameString": "fedZone1/home/rods/shared"
}
```

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<collection xmlns:ns2="http://irods.org/irods-rest">
 <ns2:metadataOperationResultEntry
  resultStatus="OK">
  <attributeString>testBulkAddCollectionAVUSendXMLAttr1</attributeString>
  <message />
  <unit>testBulkAddCollectionAVUSendXMLUnit1</unit>
  <valueString>testBulkAddCollectionAVUSendXMLValue1</valueString>
 </ns2:metadataOperationResultEntry>
 <ns2:metadataOperationResultEntry
  resultStatus="OK">
  <attributeString>testBulkAddCollectionAVUSendXMLAttr2</attributeString>
  <message />
  <unit>testBulkAddCollectionAVUSendXMLUnit2</unit>
  <valueString>testBulkAddCollectionAVUSendXMLValue2</valueString>
 </ns2:metadataOperationResultEntry>
</collection>
```

Responses - JSON


```
[
  {
    "attributeString": "testBulkAddCollectionAVUJsonAttr1",
    "valueString": "testBulkAddCollectionAVUJsonValue1",
    "unit": "testBulkAddCollectionAVUJsonUnit1",
    "resultStatus": "OK",
    "message": ""
  },
  {
    "attributeString": "testBulkAddCollectionAVUJsonAttr2",
    "valueString": "testBulkAddCollectionAVUJsonValue2",
    "unit": "testBulkAddCollectionAVUJsonUnit2",
    "resultStatus": "OK",
    "message": ""
  }
]
```

Chapter 9. Operations on Data Object ACLs (Permissions)

The following operations apply to the ACL sub-resource of iRODS data objects, and can alter access permissions.

9.1. Get Data Object Permissions

9.1.1. Description

This `GET` operation will retrieve the permissions associated with an iRODS data object

9.1.2. Requests

Syntax

```
GET /dataObject/irodsabsolutepathtodataobject.extension/acl
```

Request Parameters

Name	Description
n/a	

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:permissionListing xmlns:ns2="http://irods.org/irods-rest">
 <absolutePathString>/fedZone1/home/test1/jargon-scratch/RestCollectionServiceTest/
testGetCollectionAclXML
 </absolutePathString>
 <inheritance>true</inheritance>
 <objectType>COLLECTION</objectType>
 <permissionEntries>
  <filePermissionEnum>OWN</filePermissionEnum>
  <userId>10007</userId>
  <userName>rods</userName>
  <userType>RODS_ADMIN</userType>
  <userZone>fedZone1</userZone>
 </permissionEntries>
```

```
 <permissionEntries>
  <filePermissionEnum>OWN</filePermissionEnum>
  <userId>10012</userId>
  <userName>test1</userName>
  <userType>RODS_ADMIN</userType>
  <userZone>fedZone1</userZone>
 </permissionEntries>
</ns2:permissionListing>
```

Responses - JSON

```
{
  "permissionEntries": [
    {
      "userName": "rods",
      "userZone": "fedZone1",
      "userId": "10007",
      "userType": "RODS_ADMIN",
      "filePermissionEnum": "OWN"
    },
    {
      "userName": "test1",
      "userZone": "fedZone1",
      "userId": "10012",
      "userType": "RODS_ADMIN",
      "filePermissionEnum": "OWN"
    }
  ],
  "objectType": "COLLECTION",
  "absolutePathString": "/fedZone1/home/test1/jargon-scratch/
RestCollectionServiceTest/testGetCollectionAclJson",
  "inheritance": true
}
```

9.2. Add Data Object Permission

9.2.1. Description

This `PUT` operation will set a data object permission. Note that this is an idempotent method, and if an existing permission is already stored, the new permission will replace it. This means updating a permission is accomplished using this same method.

Note that this method returns no body, and a normal operation returns an `HTTP 204 response code`.

9.2.2. Requests

Syntax

`PUT /dataObject/irodsabsolutepathtodataobject.extension/acl/username`

Note on username iRODS supports a `user#zone` format to describe user names. This special format is useful when defining permissions on a federated grid. In normal circumstances, just the user name is required. If the zone information is also required, it should be expressed in the username portion of the url path in `username,zone` format rather than `username#zone` format. This helps clarify the URL and prevents conflicts with the URL anchor pattern.

Request Parameters

Table 9.1. Data object permission addition

Name	Description
recursive	Indicates whether the operation should be applied recursively. (`true` | `false`) The default is `false`
permission	The permission value to set (`READ` | `WRITE` | `OWN`). The default is `READ`

Responses - XML

n/a

Responses - JSON

n/a

9.3. Remove Data Object Permission

9.3.1. Description

This `DELETE` operation will remove a data object permission. Note that this is an idempotent method, and if no permission exists, it will silently ignore the request and return a normal response.

This method returns no body, and a normal operation returns an `HTTP 204 response code`.

9.3.2. Requests

Syntax

```
DELETE /dataObject/irodsabsolutepathtodataobject.extension/acl/username
```

Note on username: iRODS supports a `user#zone` format to describe user names. This special format is useful when defining permissions on a federated grid. In normal circumstances, just the user name is required. If the zone information is also required, it should be expressed in the username portion of the url path in `username,zone` format rather than `username#zone` format. This helps clarify the URL and prevents conflicts with the URL anchor pattern.

Request Parameters

Table 9.2. Data object permission deletion

Name	Description
recursive	Indicates whether the operation should be applied recursively. (`true` | `false`) The default is `false`

Responses - XML

n/a

Responses - JSON

n/a

Chapter 10. Operations on Rules

10.1. Execute an iRODS Rule by providing a rule body and any override parameters

10.1.1. Description

This `POST` operation will send a client-submitted rule to iRODS for invocation, returning back a document that contains the configured output parameters and log information.

In this case, the rule is provided in the request, and optional override parameters can be provided to specify values for iRODS rule input parameters. These are substituted in the rule processing code.

Note that iRODS rules can either be the `old` format, pre iRODS 3.2, or they can be in the updated format of the enhanced rule engine. The rule processing type will be specified in the request body along with the rule string.

10.1.2. Requests

Syntax

`POST /rule`

Request Parameters

Name	Description
n/a	

Note that the rule processing is controlled by the JSON or XML request body, as demonstrated below. The following examples show the inclusion of an override parameter. Also note that the request includes a 'ruleProcessingType' element that will be (`CLASSIC | INTERNAL | EXTERNAL`). The `CLASSIC` form is appropriate for the old rule engine format, `INTERNAL` and `EXTERNAL` are options of the new rule engine syntax.

Request - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

```
<ns2:rule xmlns:ns2="http://irods.org/irods-rest">
 <ruleAsOriginalText>HelloWorld {
  writeLine("stdout", "Hello, world!");
  }
  INPUT null
  OUTPUT ruleExecOut
 </ruleAsOriginalText>
 <ruleProcessingType>INTERNAL</ruleProcessingType>
</ns2:rule>
```

Request - JSON

```
{
  "ruleProcessingType": "INTERNAL",
  "ruleAsOriginalText": "myTestRule {\r\n# Input parameters are:\r\n#  Data
 object path\r\n#  Optional flags in form Keyword=value\r\n#    ChksumAll=
\r\n#    verifyChksum=\r\n#    forceChksum=\r\n#    replNum=\r\n# Output
 parameters are:\r\n#  Checksum value\r\n# Output from running the example
 is\r\n#  Collection is /tempZone/home/rods/sub1 and file is foo1\r\n#  Saved
 checksum for file foo1 is f03e80c9994d137614935e4913e53417, new checksum is
 f03e80c9994d137614935e4913e53417 \r\n   msiSplitPath(*dataObject,*Coll,*File);
\r\n   writeLine(\"stdout\",\"Collection is *Coll and file is *File\");\r
\n   msiMakeGenQuery(\"DATA_CHECKSUM\",\"DATA_NAME = '*File' AND COLL_NAME
 = '*Coll'\",*GenQInp);\r\n   msiExecGenQuery(*GenQInp,*GenQOut);\r\n
 foreach(*GenQOut) {\r\n     msiGetValByKey(*GenQOut,\"DATA_CHECKSUM\",*chkSumS);
\r\n     msiDataObjChksum(*dataObject,*Flags,*chkSum);\r\n     writeLine(\"stdout
\",\"Saved checksum for file *File is *chkSumS, new checksum is *chkSum
\");\r\n  }\r\n}\r\nINPUT *dataObject=\"/test1/home/test1/jargon-scratch/
RuleProcessingAOImplTest/testExecuteRuleFromResourceWithOverrides.txt\", *Flags=
\"forceChksum=true\"\r\nOUTPUT ruleExecOut\r\n",
  "irodsRuleInputParameters": [
    {
      "name": "*dataObject",
      "value": "\"/fedZone1/home/test1/jargon-scratch/RuleServiceTest/
testExecuteNewFormatRuleWithOverride.txt\""
    }
  ]
}
```

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:rule xmlns:ns2="http://irods.org/irods-rest">
 <ruleAsOriginalText>HelloWorld {
  writeLine("stdout", "Hello, world!");
  }
```

```
  INPUT null
  OUTPUT ruleExecOut
 </ruleAsOriginalText>
 <ruleProcessingType>INTERNAL</ruleProcessingType>
</ns2:rule>
```

Responses - JSON

```
{
  "outputParameterResults": [
    {
      "parameterName": "ruleExecOut",
      "outputParamType": "STRING",
      "resultObject": "Collection is \"/fedZone1/home/test1/jargon-scratch/
RuleServiceTest and file is testExecuteNewFormatRuleWithOverride.txt\"\n"
    },
    {
      "parameterName": "ruleExecErrorOut",
      "outputParamType": "STRING",
      "resultObject": ""
    }
  ]
}
```

Add user and group operations currently undocumented!

Chapter 11. Operations on Users

11.1. Obtain a temporary iRODS password for a user

11.1.1. Description

This `PUT` operation will obtain a temporary iRODS password for a user. This may be used by any user to obtain a temporary password for themselves, and may be used in admin mode by a properly authorized user to generate a temporary password for a different user.

11.1.2. Requests

Syntax

```
PUT /user/userName/temppassword
```

Request Parameters

Table 11.1. Temporary user password generation

Name	Description
admin	`true` | `false`, with `false` as the default. This indicates the request is in admin mode, allowing obtaining a temporary password for a different user than the logged in user.

Response - JSON

```
{
  "userName": "test2",
  "password": "83328ce8166933efab694b6dbeed102a"
}
```

Chapter 12. GenQuery Operations

12.1. Description

This `POST` operation performs GenQuery queries. The query's request body consists of one or more of the following fields:

- `select` - A list of fields that are to be selected. The field names must match one of the enums in `org.irods.jargon.core.query.RodsGenQueryEnum`.
- `aggregate_type` – An optional attribute for the select field. The aggregate_type must match one of the enums in `org.irods.jargon.core.query.GenQueryField.SelectFieldTypes`.
- `condition` – A list of query conditions. Each condition includes the following:
  - `column` – The query column. This must match one of the enums in `org.irods.jargon.core.query.RodsGenQueryEnum`.
  - `operator` – The operator for the conditional test. This must match one of the enums in `org.irods.jargon.core.query.QueryConditionOperators`.
  - `value` – The value the column is being tested against. This applies to all operators except the `IN` operator.
  - `value_list` – A list of values for the IN operator. The `value_list` contains one or more value fields.
- `order_by` – A list of order-by conditions. Each order-by condition includes the following:
  - `column` – The order-by column. This must match one of the enums in `org.irods.jargon.core.query.RodsGenQueryEnum`.
  - `order_condition` – The order condition. This must match one of the enums in `org.irods.jargon.core.query.OrderByType`.

12.2. Example Non Aggregate Query

The XML and JSON queries listed below are the equivalent of the following query.

```
select RESC_NAME, COLL_NAME, DATA_NAME, DATA_SIZE
where  COLL_NAME = '/tempZone/home/rods' and DATA_NAME like '%.dat'
order by DATA_SIZE DESC
```

12.2.1. Requests

Syntax

`POST /genQuery`

Request Parameters

Name	Description
n/a	

Request Body - XML

```
<ns2:query xmlns:ns2="http://irods.org/irods-rest">
  <select>RESC_NAME</select>
  <select>COLL_NAME</select>
  <select>DATA_NAME</select>
  <select>DATA_SIZE</select>
  <condition>
    <column>COLL_NAME</column>
    <operator>EQUAL</operator>
    <value>/tempZone/home/rods</value>
  </condition>
  <condition>
    <column>DATA_NAME</column>
    <operator>LIKE</operator>
    <value>%.dat</value>
  </condition>
  <order_by>
    <column>DATA_SIZE</column>
    <order_condition>DESC</order_condition>
  </order_by>
</ns2:query>
```

Request Body - JSON

```
{
  "select": [
    {
      "value": "RESC_NAME"
    },
    {
      "value": "COLL_NAME"
    },
```

```
    {
      "value": "DATA_NAME"
    },
    {
      "value": "DATA_SIZE"
    }
  ],
  "condition": [
    {
      "column": "COLL_NAME",
      "operator": "EQUAL",
      "value": "/tempZone/home/rods/"
    },
    {
      "column": "DATA_NAME",
      "operator": "LIKE",
      "value": "%.dat"
    }
  ],
  "order_by": [
    {
      "column": "DATA_SIZE",
      "order_condition": "DESC"
    }
  ]
}
```

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
  <ns2:results xmlns:ns2="http://irods.org/irods-rest">
  <row>
    <column name="RESC_NAME">demoResc</column>
    <column name="COLL_NAME">/tempZone/home/rods/</column>
    <column name="DATA_NAME">testfile3.dat</column>
    <column name="DATA_SIZE">40</column>
  </row>
  <row>
    <column name="RESC_NAME">demoResc</column>
    <column name="COLL_NAME">/tempZone/home/rods</column>
    <column name="DATA_NAME">testfile2.dat</column>
    <column name="DATA_SIZE">30</column>
  </row>
  <row>
    <column name="RESC_NAME">demoResc</column>
    <column name="COLL_NAME">/tempZone/home/rods/</column>
    <column name="DATA_NAME">testfile1.dat</column>
```

```
    <column name="DATA_SIZE">20</column>
  </row>
</ns2:results>
```

Responses - JSON

```
{
  "row": [
    {
      "column": [
        {
          "name": "RESC_NAME",
          "value": "demoResc"
        },
        {
          "name": "COLL_NAME",
          "value": "/tempZone/home/rods/"
        },
        {
          "name": "DATA_NAME",
          "value": "testfile3.dat"
        },
        {
          "name": "DATA_SIZE",
          "value": "40"
        }
      ]
    },
    {
      "column": [
        {
          "name": "RESC_NAME",
          "value": "demoResc"
        },
        {
          "name": "COLL_NAME",
          "value": "/tempZone/home/rods/"
        },
        {
          "name": "DATA_NAME",
          "value": "testfile2.dat"
        },
        {
          "name": "DATA_SIZE",
          "value": "30"
        }
      ]
```

```
    },
    {
      "column": [
        {
          "name": "RESC_NAME",
          "value": "demoResc"
        },
        {
          "name": "COLL_NAME",
          "value": "/tempZone/home/rods/"
        },
        {
          "name": "DATA_NAME",
          "value": "testfile1.dat"
        },
        {
          "name": "DATA_SIZE",
          "value": "20"
        }
      ]
    }
  ]
}
```

12.3. Example Aggregate Query

The XML and JSON queries listed below are the equivalent of the following query:

```
select SUM(DATA_SIZE)
where COLL_NAME = '/tempZone/home/rods' and DATA_NAME like '%.dat'
```

12.3.1. Requests

Syntax

`POST /genQuery`

Request Parameters

Name	Description
n/a	

Request Body - XML

```
<ns2:query xmlns:ns2="http://irods.org/irods-rest">
  <select aggregate_type="SUM">DATA_SIZE</select>
  <condition>
    <column>COLL_NAME</column>
    <operator>EQUAL</operator>
    <value>/tempZone/home/rods</value>
  </condition>
  <condition>
    <column>DATA_NAME</column>
    <operator>LIKE</operator>
    <value>%.dat</value>
  </condition>
</ns2:query>
```

Request Body - JSON

```
{
  "select": [
    {
      "value": "DATA_SIZE",
      "aggregate_type": "SUM"
    }
  ],
  "condition": [
    {
      "column": "COLL_NAME",
      "operator": "EQUAL",
      "value": "/tempZone/home/rods"
    },
    {
      "column": "DATA_NAME",
      "operator": "LIKE",
      "value": "%.dat"
    }
  ]
}
```

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:results xmlns:ns2="http://irods.org/irods-rest">
  <row>
    <column name="SUM(DATA_SIZE)">90</column>
  </row>
</ns2:results>
```

Responses - JSON

```
{
  "row": [
    {
      "column": [
        {
          "name": "SUM(DATA_SIZE)",
          "value": "90"
        }
      ]
    }
  ]
}
```

12.4. Example Using `IN` Clause with a `value_list`

The XML and JSON queries listed below are the equivalent of the following query:

```
select COLL_NAME, DATA_NAME
where COLL_NAME = '/tempZone/home/rods/dir/GenQueryTestDirectory' and DATA_NAME in
 ('test_file1.dat', 'test_file2.dat')
```

12.4.1. Requests

Syntax

`POST /genQuery`

Request Parameters

Name	Description
n/a	

Request Body - XML

```
<ns2:query xmlns:ns2="http://irods.org/irods-rest">
  <select>COLL_NAME</select>
  <select>DATA_NAME</select>
  <condition>
    <column>COLL_NAME</column>
    <operator>EQUAL</operator>
```

```
    <value>/tempZone/home/rods/dir/GenQueryTestDirectory</value>
  </condition>
  <condition>
    <column>DATA_NAME</column>
    <operator>IN</operator>
    <value_list>
      <value>testfile1.dat</value>
      <value>testfile2.dat</value>
    </value_list>
  </condition>
</ns2:query>
```

Request Body - JSON

```
{
  "select": [
    {
      "value": "COLL_NAME"
    },
    {
      "value": "DATA_NAME"
    }
  ],
  "condition": [
    {
      "column": "COLL_NAME",
      "operator": "EQUAL",
      "value": "/tempZone/home/rods/dir/GenQueryTestDirectory"
    },
    {
      "column": "DATA_NAME",
      "operator": "IN",
      "value_list": {
        "value": [
          "testfile1.dat",
          "testfile2.dat"
        ]
      }
    }
  ]
}
```

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:results xmlns:ns2="http://irods.org/irods-rest">
```

```
  <row>
    <column name="COLL_NAME">/tempZone/home/rods/dir/GenQueryTestDirectory</column>
    <column name="DATA_NAME">testfile1.dat</column>
  </row>
  <row>
    <column name="COLL_NAME">/tempZone/home/rods/dir/GenQueryTestDirectory</column>
    <column name="DATA_NAME">testfile2.dat</column>
  </row>
</ns2:results>
```

Responses - JSON

```
{
  "row": [
    {
      "column": [
        {
          "name": "COLL_NAME",
          "value": "/tempZone/home/rods/dir/GenQueryTestDirectory"
        },
        {
          "name": "DATA_NAME",
          "value": "testfile1.dat"
        }
      ]
    },
    {
      "column": [
        {
          "name": "COLL_NAME",
          "value": "/tempZone/home/rods/dir/GenQueryTestDirectory"
        },
        {
          "name": "DATA_NAME",
          "value": "testfile2.dat"
        }
      ]
    }
  ]
}
```

Chapter 13. Operations on Tickets

13.1. Create a Ticket

13.1.1. Description

This `POST` operation will create an iRODS ticket.

The input fields are:

- `mode` – either `read` or `write`
- `object_path` – the full path to the data object or collections
- `ticket_string` – the ticket's string. This is optional. If it is not provided the string will be generated by the system.

13.1.2. Requests

Syntax

`POST /ticket`

Request Parameters

Name	Description
n/a	

Request Body - XML

```
<ns2:ticket xmlns:ns2="http://irods.org/irods-rest">
    <mode>read</mode>
    <object_path>/tempZone/home/rods</object_path>
    <ticket_string>111222333444555</ticket_string>
</ns2:ticket>
```

Request Body - JSON

```
{
  "mode": "read",
  "object_path": "/tempZone/home/rods",
  "ticket_string": "111222333444555"
```

```
}
```

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ticket xmlns:ns2="http://irods.org/irods-rest">
    <ticket_string>111222333444555</ticket_string>
</ns2:ticket>
```

Responses - JSON

```
{
  "ticket_string": "111222333444555"
}
```

13.2. Delete a Ticket

13.2.1. Description

This `DELETE` operation will delete an iRODS ticket.

13.2.2. Requests

Syntax

`DELETE /ticket/ticketString`

Request Parameters

Name	Description
n/a	

13.3. Update a Ticket

13.3.1. Description

This `PUT` operation will update an iRODS ticket.

The updates has two files – `restriction_type` and `restriction_value`.

The following are the valid restriction types and values:

Table 13.1. Valid restriction types and values

restriction_type	restriction_value
`offset`	A recognizable hostname.
`remove_host`	A hostname that had been previously added.
`add_group`	An iRODS group.
`remove_group`	An iRODS group that had been previously added.
`add_user`	An iRODS user.
`remove_user`	An iRODS user that had been previously added.
`byte_write_limit`	An integer
`file_write_limit`	An integer
`uses_limit`	An integer
`expiration`	The date/time in the format yyyy-MM-dd HH:mm:ss

13.3.2. Requests

Syntax

`PUT /ticket/ticketString`

Request Body - XML

```
<ns2:ticket xmlns:ns2="http://irods.org/irods-rest">
  <restriction_type>expiration</restriction_type>
  <restriction_value>2016-02-27 13:00:00</restriction_value>
</ns2:ticket>
```

Request Body - JSON

```
{
  "restriction_type": "uses_limit",
  "restriction_value": "12"
}
```

13.4. List Ticket Information

13.4.1. Description

This `GET` operation will list the information for the selected ticket.

Syntax

```
GET /ticket/ticketString
```

Request Parameters

Name	Description
n/a	

Responses - XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:ticket xmlns:ns2="http://irods.org/irods-rest">
  <ticket_id>15542</ticket_id>
  <ticket_string>kZeGACQHEjRHFyC</ticket_string>
  <ticket_type>read</ticket_type>
  <object_type>data_object</object_type>
  <owner_name>rods</owner_name>
  <owner_zone>tempZone</owner_zone>
  <uses_count>0</uses_count>
  <uses_limit>20</uses_limit>
  <write_file_count>0</write_file_count>
  <write_file_limit>10</write_file_limit>
  <write_byte_count>0</write_byte_count>
  <write_byte_limit>1000</write_byte_limit>
  <expire_time>2017-02-07 14:18:28</expire_time>
  <irods_path>/tempZone/home/rods/dir/TicketTestDirectory/testfile.dat</irods_path>
  <host_restrictions>127.0.0.1</host_restrictions>
  <user_restrictions>rods</user_restrictions>
  <group_restrictions>rodsadmin</group_restrictions>
</ns2:ticket>
```

Responses - JSON

```
{
  "ticket_id": "15532",
  "ticket_string": "g7gwI7OK21mrWy9",
  "ticket_type": "read",
```

```
  "object_type": "data_object",
  "owner_name": "rods",
  "owner_zone": "tempZone",
  "uses_count": 0,
  "uses_limit": 20,
  "write_file_count": 0,
  "write_file_limit": 10,
  "write_byte_count": 0,
  "write_byte_limit": 1000,
  "expire_time": "2017-02-07 14:09:24",
  "irods_path": "/tempZone/home/rods/dir/TicketTestDirectory/testfile.dat",
  "host_restrictions": [
    "127.0.0.1"
  ],
  "user_restrictions": [
    "rods"
  ],
  "group_restrictions": [
    "rodsadmin"
  ]
}
```


13.5. List All Tickets

13.5.1. Description

This `GET` operation will list the information for all tickets.

Syntax

`GET /listAllTickets`

Request Parameters

Name	Description
n/a	

Responses - XML


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns2:tickets xmlns:ns2="http://irods.org/irods-rest">
  <ticket>
    <ticket_id>15542</ticket_id>
    <ticket_string>kZeGACQHEjRHFyC</ticket_string>
    <ticket_type>read</ticket_type>
```

```
    <object_type>data_object</object_type>
    <owner_name>rods</owner_name>
    <owner_zone>tempZone</owner_zone>
    <uses_count>0</uses_count>
    <uses_limit>20</uses_limit>
    <write_file_count>0</write_file_count>
    <write_file_limit>10</write_file_limit>
    <write_byte_count>0</write_byte_count>
    <write_byte_limit>1000</write_byte_limit>
    <expire_time>2017-02-07 14:18:28</expire_time>
    <irods_path>/tempZone/home/rods/dir/TicketTestDirectory/testfile.dat</
irods_path>
    <host_restrictions>127.0.0.1</host_restrictions>
    <user_restrictions>rods</user_restrictions>
    <group_restrictions>rodsadmin</group_restrictions>
  </ticket>
  <ticket>
    <ticket_id>15543</ticket_id>
    <ticket_string>111222333444555</ticket_string>
    <ticket_type>read</ticket_type>
    <object_type>data_object</object_type>
    <owner_name>rods</owner_name>
    <owner_zone>tempZone</owner_zone>
    <uses_count>0</uses_count>
    <uses_limit>20</uses_limit>
    <write_file_count>0</write_file_count>
    <write_file_limit>10</write_file_limit>
    <write_byte_count>0</write_byte_count>
    <write_byte_limit>1000</write_byte_limit>
    <expire_time>2017-02-07 14:18:28</expire_time>
    <irods_path>/tempZone/home/rods/dir/TicketTestDirectory/testfile.dat</
irods_path>
    <host_restrictions>127.0.0.1</host_restrictions>
    <user_restrictions>rods</user_restrictions>
    <group_restrictions>rodsadmin</group_restrictions>
  </ticket>
</ns2:tickets>
```

Responses - JSON

```
{
  "ticket": [
    {
      "ticket_id": "15532",
      "ticket_string": "g7gwI7OK21mrWy9",
      "ticket_type": "read",
      "object_type": "data_object",
```

```
      "owner_name": "rods",
      "owner_zone": "tempZone",
      "uses_count": 0,
      "uses_limit": 20,
      "write_file_count": 0,
      "write_file_limit": 10,
      "write_byte_count": 0,
      "write_byte_limit": 1000,
      "expire_time": "2017-02-07 14:09:24",
      "irods_path": "/tempZone/home/rods/dir/TicketTestDirectory/testfile.dat",
      "host_restrictions": [
        "127.0.0.1"
      ],
      "user_restrictions": [
        "rods"
      ],
      "group_restrictions": [
        "rodsadmin"
      ]
    },
    {
      "ticket_id": "15533",
      "ticket_string": "111222333444555",
      "ticket_type": "read",
      "object_type": "data_object",
      "owner_name": "rods",
      "owner_zone": "tempZone",
      "uses_count": 0,
      "uses_limit": 20,
      "write_file_count": 0,
      "write_file_limit": 10,
      "write_byte_count": 0,
      "write_byte_limit": 1000,
      "expire_time": "2017-02-07 14:09:24",
      "irods_path": "/tempZone/home/rods/dir/TicketTestDirectory/testfile.dat",
      "host_restrictions": [
        "127.0.0.1"
      ],
      "user_restrictions": [
        "rods"
      ],
      "group_restrictions": [
        "rodsadmin"
      ]
    }
  ]
}
```