Preliminary
OASIS/ebXML Registry Test Plan
and Test Requirements

by

Len Gallagher
NIST-ITL-SDCT
Conformance Testing Group

Draft - June 20, 2002

- Abstract -

The OASIS/ebXML Registry Information Model (RIM) and Registry Services (RS) specifications were approved by OASIS membership for adoption as OASIS standards in May 2002. The RIM specification uses UML to define abstract structures for holding registry information and abstract methods for registry behavior. There is no conformance requirement to implement these structures and methods as specified; instead, one claims conformance to the information model of RIM and to the services defined by RS. The RS specification provides an XML services interface for invoking the RIM methods. The specifications are a pair and conformance to RIM alone does not guarantee interoperability. This test plan focuses on the XML requests and responses defined by the RS specification. This document presents a test plan and high level requirements that will enable subsequent development of lower level test requirements, specific test assertions, and executable test cases. An implementation wishing to claim conformance to the OASIS/ebXML Registry specification shall execute all test cases successfully. These test requirements and all referenced test assertions and test cases are provided by NIST as a public service. All referenced files are freely accessible in a public NIST FTP server located at URL ftp://xsun.sdct.itl.nist.gov/regrep.

- Table of Contents -

1. Introduction

The OASIS/ebXML Registry Information Model (RIM) defines the classes and methods of a registry information model. The OASIS/ebXML Registry Services Specification (RS) defines integrated XML schemas to represent classes of RIM, submission and update services of RS, and query services of RS. The submission, update, and query services of RS constitute an XML services interface to the structures and methods defined by RIM. Both specifications and related XML schema definition documents are available from OASIS as follows:

OASIS/ebXML Registry Information Model v2.0, 18 December 2001, 60 pages
http://www.ebxml.org/specs/ebrim2.pdf

OASIS/ebXML Registry Services Specification v2.0, 6 December 2001, 128 pages
http://www.ebxml.org/specs/ebrs2.pdf

XML Schema for Information Model, 8 pages
http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rim.xsd

XML Schema for Life Cycle Registry Services, 4 pages
http://www.oasis-open.org/committees/regrep/documents/2.0/schema/rs.xsd

XML Schema for Query Registry Services, 9 pages
http://www.oasis-open.org/committees/regrep/documents/2.0/schema/query.xsd

We envision an ebXML Registry Test Suite consisting entirely of XML instance documents that validate to the XML schema definitions in RS. In Section 6.2, the RS specification defines an abstract ebXML Registry Service to consist of two key functional interfaces called QueryManager and LifeCycleManager. Each interface is defined by a collection of XML requests and XML responses. In Section 6.3, the RS specification defines two alternative concrete bindings for the abstract registry service:

- A SOAP binding using the HTTP protocol, or
- An ebXML Messaging Service binding defined as a separate OASIS/ebXML standard.

Our test plan and test requirements focus almost entirely on the QueryManager and LifeCycleManager services, thereby allowing the test suite to be bound to any messaging service. When absolutely necessary, we will assume the simplest possible SOAP implementation to carry an XML request from a registry client to a registry server and to return an XML response from the server back to the client.

The test suite can be used in several different ways: [DISCUSS THE FOLLOWING ALTERNATIVES]
- Manual conformance testing with direct submission of XML requests to an implementation and manual analysis of the returned responses.
- A Test Harness to automatically do the submission of requests and analysis of responses.
- A Reference Implementation interaction with an implementation under conformance testing.
- Interoperability testing among 2 or more supposedly conforming implementations.

The overall test plan consists of the following steps:

- Test setup – a set of XML requests to define new Organization instances and new User instances for each organization. Each subsequent request in the test suite is invoked by one of these users on behalf of one of these organizations.

- Structure population – a set of XML requests to populate the core classes of the Registry information model. Each request will be followed by a simple query to determine if the request was acted upon properly by the Registry implementation.

- Maintenance testing – a set of XML requests to modify the state of existing objects in the Registry. Each request will be followed by a simple query to determine if the request was acted upon properly by the Registry implementation. Some requests may also require submission of new object instances to put the Registry into a proper state for testing a specific maintenance request.

- Query testing – a set of XML requests to execute some of the more sophisticated query requirements specified by RS. Some requests will be executed against the Registry instances created by the preceding structure population, but others may require additional LifeCycle requests to modify instances or add new instances to put the Registry into the proper state for testing a specific query. Each test will leave the Registry in the same state as it was before the initial request of that test so that this section of the test suite will be independent of the order of execution of the individual test cases.

Each of these steps is described more fully in the remaining sections below.

2. Test setup and initialization

The purpose of this step is to create a small number of Organization and User instances that will be used for subsequent test requests. According to RIM Section 8.2, each user is linked to exactly one organization and makes requests on behalf of that organization. The organization, not the individual user, will be the owner of any objects submitted. In Section 8.3.4, RIM requires that exactly one user for each organization is the primaryContact for that organization. Although not explicitly required by RIM, these test requirements assume that the primaryContact for each organization is a user whose organization attribute links back to that same organization. According to the UML diagram of Figure 1 in RIM Section 6, an organization may have multiple users.

An organization may be the parent of another organization, as anticipated by the *parent* attribute of the Organization class defined in RIM Section 8.3.3. The specification is clear that an organization and its parent organization act separately in all situations and ownership of submitted objects is not inferred by one from the other. However, the specification is open to interpretation in deciding when an existing organization can be referenced as the parent of a newly submitted organization. See Section 2.5 below for further discussion of the parent/child relationship for organizations.

In Section 5.3, RS describes the different types of Registry users, including Registered User and Registry Guest. A Registry Guest does not have to be authenticated for Registry access, but only a Registered User can submit or update Registry content. Although not explicitly required by RS Section 5.3, these test requirements assume that every Registered User is represented by a User instance in the information model. RS Section 5.3 also defines a Submitting Organization to be a Registered User, which would require that the same identity be an instance of both the User and Organization classes. These test requirements assume that when RS references a Submitting Organization as a Registered User, it really means a user who is authorized to act on behalf of that organization.

At the end of the setup and initialization step, this test plan assumes the existence of Organization and User instances with name attributes as follows:

Organization name	User name
Registration Authority	RAprimary
SubmitOrgFrequent	SOFprimary
SubmitOrgFrequent	SOFsecondary
SubmitOrgParent	SOPprimary
SubmitOrgChild	SOCprimary
ResponsibleOrg	ROprimary

Since there is a one-to-one correspondence between these organization and primary user instances, they will have to be created at the same time. The intent of these test requirements is that the primary user of the Registration Authority organization is the Registry Administrator. Only the Registration Authority or Registry Administrator may set the values of attributes that are not mutable. The other four organizations are treated as normal Organization instances whose users will make requests for registry services

The following subsections address test requirements for establishing the organizations and users identified above.

2.1 Communication bootstrapping

Read RS Section 6.6.2.1 page 23 to see what must be done. Note that every Registry claiming conformance with a SOAP binding shall provide a WSDL description. However, for ebXML bootstrapping a WSDL is NOT required. What are the test requirements here??

2.2 Registry Users

Read RS Section 5.3! Line 347 says that "Registration of a user happens out-of-band, i.e. by a means not specified by this specification." In spite of this statement in RS, there will be SubmitObjects test assertions to submit name and address information for the five organizations and six users listed above. In order to be successful for these tests, the Registry implementation must accept those requests and create appropriate Organization and User instances. However, the means by which the Registration Authority is distinguished from the other organizations and the means by which authentication information for users is created or communicated back to the request is implementation defined.

2.3 Registration authority

The Registration Authority (RA) plays a special role in both RIM and RS and it is not clear from the specifications if that special role must be visible as an organization to other users. In RS, the Registration Authority also plays the role of Registry Administrator, cf Section 5.3.

Test Requirements

1. Allow creation of a Registration Authority for this Registry by submission of an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of exactly one XML Organization element and exactly one XML User element. The primaryContact attribute of Organization references the user and the organization attribute of User references the organization. All other attributes are minimal. The XML SubmitObjectsRequest for this action is at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRA.xml.
2. The organization identified by the XML Organization element will be the Registration Authority for the duration of all test cases.
3. The user identified by XML User element will be the Registry Administrator for the duration of all test cases. This user may have a special interface to the Registry not allowed to any other user.
4. The Registration Authority and the Registry Administrator may or may not be visible to other users through queries on Organization and/or User classes.

2.4 Submitting organizations

The intent is to create two potential submitting organizations with the following names: SubmitOrgFrequent, and SubmitOrgParent. The majority of subsequent test cases will be submitted by the primaryContact of the SubmitOrgFrequent. In the following section SubmitOrgParent will establish another organization SubmitOrgChild that refers back to SubmitOrgParent as its parent organization.

Test Requirements

1. Allow creation of multiple potential Submitting Organizations for this test Registry. Initially, the tests will make two separate submissions of an XML SubmitObjectsRequest. The LeafRegistryObjectList of each request consists of exactly one XML Organization element and exactly one XML User element. In each case, the primaryContact attribute of Organization references the user and the organization attribute of User references the organization.
2. The first SubmitObjectsRequest has the following information for Organization: 1 Name "SubmitOrgFrequent", 1 Description in English, 0 Slots, 0 Classifications, 0 ExternalIdentifiers, 0 accessControlPolicy, 1 id "SOF", 0 objectType, 1 Address with all non-empty attributes and a Slot for "Mail Stop", 2 TelephoneNumbers with all non-empty attributes, 0 parent, 1 primaryContact "SOFPC"; and the following information for User: 1 Name "SOFprimary", 1 Description in English, one Slot for "Test Suite Roles" with 3 values, 0 Classifications, 0 ExternalIdentifiers, 0 accessControlPolicy, id "SOFPC", 0 objectType, 1 Address only one non-empty attribute, 1 PersonName with non-empty firstName and lastName, 2 TelephoneNumbers with various non-empty attributes, 1 emailAddress, 1 organization "SOF", 0 url. The XML SubmitObjectsRequest for this action is at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOF.xml. NOTE: The tool used here for XML instance creation is saying that the "lang" attribute on the LocalizedString sub-element of Description is an undeclared attribute. Is this an error in the tool or an error in the RIM XML schema?

3. The second SubmitObjectsRequest has the following information for Organization: 1 Name “SubmitOrgParent”, 1 Description in English, 0 Slots, 0 Classifications, 0 ExternalIdentifiers, 0 accessControlPolicy, 1 id “SOP”, 0 objectType, 1 Address, 3 TelephoneNumbers, 0 parent, 1 primaryContact “SOPPC”; and the following information for User: 1 Name “SOPprimary”, 1 Description with LocalizedString instances in both English and French, 0 Slots, 0 Classifications, 0 ExternalIdentifiers, 0 accessControlPolicy, id “SOPPC”, 0 objectType, 1 Address, 1 PersonName with all empty attributes, 1 TelephoneNumber with various non-empty attributes, 1 emailAddress with non-empty address, 1 organization “SOP”, 0 url. The XML SubmitObjectsRequest for this action is at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOP.xml.
4. How does the test suite determine the 128bit UUID generated by the Registry for each organization and each user? Is it retuned in any way by the SubmitObjectsRequest or is it necessary to submit a query to identify objects of interest?

2.5 Child organization

The intent here is to create a new organization that references an existing organization as its parent. Neither RIM nor RS states any constraints on which existing organizations can be referenced as the parent of a proposed new organization. The specifications are also silent on whether or not a user of an existing organization can create a new organization that references the existing organization as its parent. Subsequent tests will assume the existence of a child organization that references a parent organization, with a new primaryContact, whether the child organization is submitted by an existing user of the proposed parent organization or not.

Test Requirements

1. Determine the 128 bit UUID generated by the Registry for SubmitOrgParent, call it SOP128bitUUID. The XML AdhocQueryRequest for this action is at URL ftp://xsun.sdct.itl.nist.gov/regrep/query/FindSOP.xml. The single ObjectRef returned in the AdhocQueryResponse will have an id attribute whose value is the desired UUID.
2. Allow submission of a SubmitObjectsRequest for creation of a new Organization instance that references SubmitOrgParent as its parent organization. The LeafRegistryObjectList consists of exactly one XML Organization element and exactly one XML User element. The primaryContact attribute of Organization references the user and the organization attribute of User references the organization. Note: There may be some constraints on which users are allowed to make such a request (e.g. only the primaryContact of SubmitOrgParent). The SubmitObjectsRequest has the following information for Organization: 1 Name “SubmitOrgChild”, 1 Description in English, 0 Slots, 0 Classifications, 0 ExternalIdentifiers, 0 accessControlPolicy, 1 id “SOC”, 0 objectType, 1 Address, 1 TelephoneNumber, 1 parent “SOP128bitUUID”, 1 primaryContact “SOCPC”; and the following information for User: 1 Name “SOCprimary”, 1 Description in English, 0 Slots, 0 Classifications, 0 ExternalIdentifiers, 0 accessControlPolicy, id “SOCPC”, 0 objectType, 1 Address with all null attributes, 1 PersonName with non-empty firstName, 1 TelephoneNumber with all null attributes, 1 emailAddress, 1 organization “SOC”, 0 url. The XML SubmitObjectsRequest for this action is at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOC.xml, but the placeholder SOP128bitUUID must be replaced by the actual UUID before execution.

2.6 Responsible organization

As far as RIM and RS are concerned a responsible organization is no different from any other organization, except that it may be referenced as the targetObject of some Association instance with associationType equal to *ResponsibleFor*. There is no special role specified for responsible organizations in either specification other than the *ResonsibleFor* associationType identified in RIM Section 9.9.2 and there is no standard way to determine if an organization is allowed or not allowed to be referenced as the target of some such association. The intent here is to create a normal Organization instance with name equal to “ResponsibleOrg” and then in later test cases to identify this organization as a Responsible Organization in some new association test case submissions.

Test Requirements

1. Allow submission of a SubmitObjectsRequest for creation of an Organization instance that will later be used as the targetObject in a *ResponsibleFor* association. Even though this organization may never make any submissions to the Registry, it is required to have a primaryContact that is a User instance. The tests will attempt to create that user with a minimum of information.

2. The SubmitObjectsRequest will have a LeafRegistryObjectList that consists of exactly one XML Organization element and exactly one XML User element. The primaryContact attribute of Organization references the user and the organization attribute of User references the organization. The Organization element has the following content: 1 Name "ResponsibleOrg", 1 Description in English, 0 Slots, 0 Classifications, 0 ExternalIdentifiers, 0 accessControlPolicy, 1 id "RORG", 0 objectType, 1 Address, 1 TelephoneNumber, 0 parent, 1 primaryContact "RORGPC"; and the following information for User: 1 Name "ROprimary", 1 Description in English, 0 Slots, 0 Classifications, 0 ExternalIdentifiers, 0 accessControlPolicy, id "RORGPC", 0 objectType, 0 sequence (for Address, PersonName, TelephoneNumber, EmailAddress), 1 organization "RORG", 0 url. The XML SubmitObjectsRequest for this action is at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRORG.xml.
3. Later, tests in the Maintenance section may attempt to delete this user to see what happens! Should get an error since each organization is required to have a primaryContact.

2.7 Additional users

The RIM and RS specifications are not clear about how and under what constraints additional users may be added for a given organization. However, it is imperative that at least one user (the primaryContact?) for an organization be able to create and edit other users for that organization. In the Maintenance testing section below, there will be additional tests to see if any user from that organization can add additional users. Several test assertions below will assume that at least one organization has multiple users created by the primaryContact for an organization.

Test Requirements

1. Determine the authentication method for the Registry user with name equal to SOFprimary. Note: The identity of that user for authentication purposes may or may not be the same as its name attribute from the User class!
2. Determine the 128 bit UUID generated by the Registry for SubmitOrgFrequent, call it SOF128bitUUID. The URL ftp://xsun.sdct.itl.nist.gov/regrep/query/FindSOF128bitUUID.xml finds the XML AdhocQueryRequest. The single ObjectRef returned in the AdhocQueryResponse will have an id attribute whose value is the desired UUID.
3. Create a second user for SubmitOrgFrequent by having SOFprimary submit a SubmitObjectsRequest that consists of exactly one User instance with the following content: 1 Name "SOFsecondary", 1 Description in English, 0 Slots, 0 Classifications, 0 ExternalIdentifiers, 0 accessControlPolicy, 0 id, 0 objectType, 1 Address, 1 PersonName with non-empty firstName and lastName, 3 TelephoneNumbers with various combinations of attributes, 2 EmailAddresses with various non-empty attributes, 1 organization "SOF128bitUUID", 0 url. The XML SubmitObjectsRequest for this action is at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOF2nd.xml.

3. Structure population and verification

The purpose of this step is to populate a set of core structures with enough data to support the major requirements of subsequent maintenance testing and query testing. Completion of the above setup and initiation step ensures that the Organization and User classes have been populated with adequate organization and user instances for subsequent testing. This step will create required and useful ClassificationScheme instances, and will populate the following core classes with at least a small collection of objects useful for a majority of subsequent test cases:

- RegistryObject, cf RIM Section 7.4
- RegistryEntry as a subtype of RegistryObject, cf RIM Section 7.5
- ExtrinsicObject as a subtype of RegistryEntry, cf RIM Section 7.7
- Classification as a subtype of RegistryObject, cf RIM Section 10.3
- Association as a subtype of RegistryObject, cf RIM Section 9.9
- ExternalIdentifier as a subtype of RegistryObject, cf RIM Section 7.9
- ExternalLink as a subtype of RegistryObject, cf RIM Section 7.10
- Slot as a dependent class with instances linked to RegistryObject, cf RIM Section 7.6
- RegistryPackage as a subtype of RegistryEntry, cf RIM Section 7.8
- Service as a subtype of RegistryEntry, cf RIM Section 8.8

Objects from the other classes defined in RIM will show up as associated instances linked to the above, or as special test cases included in maintenance or query testing.

Section 7.5.6 of RIM specifies pre-defined status values for the status attribute of any RegistryEntry instance. These include *Submitted*, *Approved*, *Deprecated*, and *Withdrawn*. RIM Section 7.5.6.1 specifies that when an object is first submitted it will have a status value of *Submitted* and subsequently it will be *Approved*. RIM is silent on the process required for a repository item to move from *Submitted* to *Approved* status. Presumably the approval could be done by the Registration Authority, by the owner of the submitted item after checking that it was instantiated properly, by a separate user or users with review and approval authority, or by some combination of all of these. In Section 7.7 RS provides an Approve Objects Protocol that allows a "client" to approve one or more previously submitted repository items, but it doesn't say if the client is the RA, the owner, or some other user. RIM is silent on which users can "see" unapproved items but RS Section 7.7, line 987, implies that items must be approved before they will become available for use by business parties. It is open to interpretation whether all RegistryEntry instances require approval before they become available, e.g. ClassificationScheme and RegistryPackage, or if just *repository items* submitted along with an ExtrinsicObject instance require approval before they can be retrieved with a GetContentRequest.

These test requirements assume that ultimate approval authority resides with the RA, and that the RA automatically grants the approval privilege to the owner of each new object. Test requirements also assume that owners of un-approved registry entries can set up associations of those entries with other registry objects before approval; otherwise, it would not be possible to submit a collection of related objects and see what the effect is before final approval. In some situations below, the owner of a submitted registry entry will approve it immediately after checking that it was instantiated properly. In other cases, approval will be deferred until Section 4 as part of a general approval and confirmation process.

These test requirements use a subset of the RosettaNet Techncial Dictionary (RNTD v1.4) as the basis of a number of tests. A zip file with all of the information used herein is available from RosettaNet at the following URL:
http://www.rosettanet.org/rosettanet/Doc/0/2EF55NDSU19KF3580VHUMO6J69/RNTD+V+1.4+release.zip
Apologies to RosettaNet for any errors introduced when converting from spreadsheet representation to RegistryObject instances.

Each test requirement may yield test assertions that consist of the following steps:

- Submission of an XML SubmitObjectsRequest by a User instance created in Section 2 above.
- A check that the proper SubmitObjectsResponse is received.
- Identification of the UUID's created by the Registry for each stand-alone submitted object.
- A simple query to determine if each submitted object is properly instantiated by the Registry.
- Possible submission of an XML ApproveObjectsRequest for some submitted registry entries. Other approvals may be deferred until later actions.
- Possible confirmation of some multi-owner associations. Other confirmations may be deferred until later actions.

The following subsections address requirements for creating and verifying instances for each of the identified classes.

3.1 Pre-defined classification schemes

The RIM specification requires that certain enumeration domains be treated as if they were classification schemes. RIM Section 8.1.2.1 requires that the pre-defined auditable event types be supported by the Registry as a specific ClassificationScheme instance with name "EventType", and RIM Section 7.9.2 requires that each ExternalIdentifier instance reference a pre-existing classification scheme. Although there is no requirement that other enumerations be explicitly instantiated as classification schemes, it will be convenient for testing purposes to have those enumeration values readily available as internal or external classification schemes.

Test Requirements

1. Allow user "RAprimary" to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of one XML ClassificationScheme element with name "EventType" (as specified by RIM!) and with XML ClassificationNode elements to identify each of the pre-defined auditable event types listed in RIM Section 8.1.2.1. The SubmitObjectsRequest can be found at ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitEventTypeCS.xml.
2. Allow user "RAprimary" to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of exactly one XML ClassificationScheme element with name "External Identifier Identification Schemes" and with

several ClassificationNode elements. The intent is that this be the classification scheme whose existence is required by RIM Section 7.9.2 in any ExternalIdentifier element. It will define “ebXML”, “DUNS”, “RosettaNet”, “UUID”, “URI”, “UN/CEFACT” and “USGOVT” as possible external identifier types. Some will have uniquely named subtypes. In these test requirements, the name attribute of the ExternalIdentifier will be used in a consistent way to identify the “type” of external identifier, e.g. name = “DUNS” to indicate that the identifier is a DUNS number, or name = “RosettaNet” to indicate that the external identifier value will be unique within the RosettaNet sphere of influence. The SubmitObjectsRequest can be found at ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitEXTIDCS.xml.

3. Allow user “SOFprimary” to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of one XML ClassificationScheme element with name “RIM Pre-defined Object Types” and with XML ClassificationNode elements to identify each of the named object types listed in RIM Section 7.4.6.1. The SubmitObjectsRequest can be found at ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRIMObjectTypeCS.xml.
4. Allow user “SOFsecondary” to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of one XML ClassificationScheme element with name “RIM Pre-defined Stability Types” and with XML ClassificationNode elements to identify each of the pre-defined stability values listed in RIM Section 7.5.5.1. The SubmitObjectsRequest can be found at ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRIMStabilityTypeCS.xml.
5. Allow user “SOPprimary” to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of one XML ClassificationScheme element with name “RIM Pre-defined Status Types” and with XML ClassificationNode elements to identify each of the pre-defined status values listed in RIM Section 7.5.6.1. The SubmitObjectsRequest can be found at ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRIMStatusTypesCS.xml.
6. Allow user “SOCprimary” to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of one XML ClassificationScheme element with name “RIM Pre-defined Association Types” and with XML ClassificationNode elements to identify each of the pre-defined association types listed in RIM Section 9.9.2.1. The SubmitObjectsRequest is at ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRIMAssocTypesCS.xml.
7. Allow user “SOFprimary” to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of one XML ClassificationScheme element with name “RIM Data Types”, to be registered as an external classification scheme, and with an XML ExternalLink to a graphic that lists each of the data types discussed in RIM Section 7.2. The external link URL is ftp://xsun.sdct.itl.nist.gov/regrep/links/RIMDataTypeTable.pdf and the SubmitObjectsRequest is at ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRIMDataTypesCS.xml.

3.2 Useful classification schemes

In Section 10.1.3, the RIM specification defines three different types of classification schemes, including *UniqueCode*, *EmbeddedPath*, and *NonUniqueCode*. The following test requirements attempt to install a robust instance of each type using well-known schemes popular in electronic commerce.

The United Nations Standard Products and Services Codes (UNSPSC) is an example of an *EmbeddedPath* classification scheme with approximately 10,000 nodes in 4 levels. Each individual node has a 2-to-8 digit numeric code that completely identifies the entire path in the taxonomy. See ftp://xsun.sdct.itl.nist.gov/regrep/scheme/unspsc.txt for a non-standard XML representation of the entire UNSPSC v4.0).

The RosettaNet Techncial Dictionary (RNTD) is an example of a *NonUniqueCode* classification scheme for a group of XJA-related electronic components, with approximately 350 nodes in up to 4 levels. The code for each node is a short phrase, and the name of each node is a concatenation of the phrases (separated by a hyphen) from the first level to the level of the node. See ftp://xsun.sdct.itl.nist.gov/regrep/scheme/rntdcs.xml for a non-standard XML representation of a portion of the RosettaNet Technical Dictionary v1.4 classification scheme.

We still need an interesting example of a *UniqueCode* classification scheme. In the absence of a good example from electronic commerce we could use a hierarchy of data types, Number vs String vs Logical at level 1; Real, Complex at level 2 below Numeric; Binary, Char at level 2 below String; Boolean, 3-value at level 2 below Logical; Decimal, Float at level 3 below Real, etc. Make sure that the 5 different RosettaNet PropertyDomain types (Boolean, Integer, Real Number, String, Value Codes) are representable in the scheme. See ftp://xsun.sdct.itl.nist.gov/regrep/scheme/dtcs.xml for a non-standard XML representation of the Data Type Hierarchy classification scheme. This scheme is defined with Dynamic stability so that later in the maintenance section we can attempt to re-structure the node hierarchy and test to see if queries on existing classifications properly reflect the re-structuring.

Test Requirements

1. Allow user “RAprimary” to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of one XML ClassificationScheme element with name “UNSPSC Classification Scheme” and all 10,000+ ClassificationNode elements to completely implement the *EmbeddedPath* UNSPSC classification scheme internally to this Registry. See ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitUNSPSC.xml for the SubmitObjectsRequest.
2. Allow user “SOFprimary” to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of exactly one XML ClassificationScheme element, with name “RosettaNet Technical Dictionary Classification Scheme”, and no other elements. This should create an empty classification scheme. See ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRNTDCSscheme.xml for the SubmitObjectsRequest.
3. Allow user “SOFsecondary” to determine the UUID of the RosettaNet Technical Dictionary Classification Scheme, call it RNTDCS128bitUUID, and the UUID of the External Identifier Identification Schemes classification scheme, call it EXTIDCS128bitUUID.
4. Allow user “SOFsecondary” to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists exclusively of 350+ ClassificationNode elements to link to the existing RosettaNet Technical Dictionary Classification Scheme, and whose leaf nodes carry the class external identifier assigned by RNTD. See ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRNTDCSnodes.xml for the SubmitObjectsRequest. The two UUID identifiers for RNTDCS and EXTIDCS must be replaced by their actual UUID’s.
5. Allow user “SOCprimary” to submit an XML SubmitObjectsRequest whose LeafRegistryObjectList consists of one ClassificationScheme element and a partial collection of ClassificationNode elements from the Data Type Hierarchy classification scheme. . See ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitDTCSscheme.xml for the SubmitObjectsRequest.
6. Allow user “SOCprimary” to determine the UUID’s of a number of existing classification nodes from the previously created Data Type Hierarchy classification scheme. Let the UUID’s for the nodes for Number, Logical, and String be named DTCS:NBR128bitUUID, DTCS:LOG128bitUUID, and DTCS:STG128bitUUID, respectively.
7. Allow user “SOCprimary” to add new classification nodes as sub-nodes under existing nodes from the Data Type Hierarchy classification scheme defined above. Substitute the real UUID’s for the String, Logical, and Number nodes identified in the previous step. . See ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitDTCSnodes.xml for the SubmitObjectsRequest.

3.3 Extrinsic objects

The purpose of this section is to populate the registry with a reasonable number of *repository items*, each described by an ExtrinsicObject instance. Section 7.4.6.1 of RIM pre-defines 7 different object types for extrinsic objects. However, there is no requirement that submitted objects reference one of these pre-defined types; instead, RS defines a RegistryObject XML schema attribute for objectType as having an arbitrary string data type.

Some repository items will be text files and some will be binary files. There is no Registry requirement that the Registry implementation be able to open up the submitted *repository item* to validate it in any way, so it is not necessary for these test requirements to submit and test all possible pre-defined object types.

Applications of Registry implementations may expand the approval process to require that submitted repository items be only from a select list of objectTypes and to include additional requirements that repository items validate to an appropriate schema for each objectType. However, such enhancements are beyond the scope of the required capabilities of a Registry implementation and these test requirements make no attempt to validate submissions of *repository items*.

The repository item described by an ExtrinsicObject instance will be attached to the message along with the incoming SubmitObjectsRequest that contains the ExtrinsicObject element. A SubmitObjectsRequest may contain any number of ExtrinsicObject elements. The manner of associating an attached repository item with the appropriate ExtrinsicObject element is dependent upon the SOAP or ebXML communication binding identified in Section 2.1 above. NOTE: this is one of the few situations where Registry test requirements are dependent upon the communication service.

These test requirements adopt the convention that when a *repository item* is to be included with a submitted ExtrinsicObject element, then the ID attribute of ExtrinsicObject will be non-null. The value of that ID (call it

IDVALUE) will be placed in the Content-ID Header Field of the message to identify the attached repository item in the messaging service (i.e. SOAP or ebXML MS). The associated repository item is available in the NIST FTP directory as URL ftp://xsun.sdct.itl.nist.gov/regrep/items/IDVALUE.xxx, where "xxx" may vary depending on the type of file.

A subset of the RosettaNet technical dictionary properties will provide a number of the submitted items. We have defined an XML schema to capture the information associated with 746 property instances and have re-structured the property information so that it can be submitted to the test registry as a repository item along with its ExtrinsicObject instance. Each property has a unique 6 or 7 character code that serves as its identifier. Each will have an objectType of "RosettaNet Property". The ExtrinsicObject instance for each property will have a "ValidatesTo" association that points to the XML schema, a Classification instance to map to the data type used to represent property values and an ExternalIdentifier instance to map to its unique RosettaNet property code.

This section also includes test requirements to allow registration of certain electronic components. The intent is that these components will be classified initially by the UNSPSC classification scheme, then virtually linked to one or more corresponding RosettaNet Technical dictionary classification scheme nodes, and then virtually linked to appropriate RosettaNet properties for electronic components having that classification. Finally, in the Slot section below, test requirements will associate electronic components with a set of appropriate RosettaNet properties and supply a component value for each such property. [See file "ElecCompItemTemplate.txt"]

The first step in this test process is to register a collection of electronic components. The test requirements below take advantage of the RIM requirement specified in Section 7.4.4, lines 579-580, that a conforming implementation must accept a validly formatted UUID for a RegistryObject if one is supplied by the client. This requirement is reinforced by RS Section 7.3.1, lines 648-657, which specifies that if an id attribute in a SubmitObjectsRequest takes the form of a UUID URN, then the Registry must honor that UUID and use it as the id attribute of the object in the Registry.

Test Requirements

1. Allow user "SOFprimary" to register a single XML Schema to which all RosettaNet properties will validate. See ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitUNSPSC.xml for the SubmitObjectsRequest, and see ftp://xsun.sdct.itl.nist.gov/regrep/dtd/RosettaNetProperty.xsd for the associated repository item.
2. Allow user "SOFprimary" to register 746 distinct RosettaNet properties, each with objectType = "RosettaNetProperty", each classified by its data type representation, each with an external identifier to hold its unique RosettaNet property code, and each with an association of type *ValidatesTo* that links the property to its defining XML Schema. The user must first determine the test Repository UUID's for the following already submitted items: the ExternalIdentifier classification scheme (EXTIDCS128bitUUID), the RosettaNet XML schema for properties (RNPXSD128bitUUID), and the following RosettaNet data type nodes from the classification scheme for data types: Integer (DTCS:INT128bitUUID), Real Number (DTCS:DEC128bitUUID), Boolean (DTCS:BOOL128bitUUID), String (DTCS:STRG128bitUUID), and enumeration of codes (DTCS:CODE128bitUUID). The SubmitObjectsRequest for this action is at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRNProperties.xml; it will have to be modified to make substitutions for these 7 UUID's. The associated repository items are in the NIST FTP directory located at URL ftp://xsun.sdct.itl.nist.gov/regrep/items/ each with a file name equal to "XXXXXX.xml" where "XXXXXX" is the RosettaNet property code.
3. In the previous step, each ExtrinsicObject instance was submitted with status = "Approved". The RIM and RS specifications are not clear as to whether or not a conforming implementation must accept this status value suggested by the client; instead, some conforming implementations may require that all submittals be accepted first with status = "Submitted" and later be explicitly modified to status = "Approved" via the explicit Approve Objects Protocol specified in RS. The purpose of this test requirement is to ensure that the Registry Administrator take whatever steps are needed to change the status of each RosettaNetProperty submitted above to have status = "Approved". This means that users other than SOFprimary should be able to reference and access these properties as necessary in subsequent test requirements.
4. Upon completion of the above test requirement there will be 746 RosettaNet properties registered in this test Registry. In subsequent tests, we will need to know the UUID's that were assigned to these properties by the Registry under test. Let XXXXXX_128bitUUID be the 128 bit UUID that was assigned to the RosettaNet property with property code equal to XXXXXX. In subsequent test cases we use this identifier in place of the actual UUID. A substitution will have to be made to replace each such identifier with its actual UUID before execution. The AdhocQueryRequest at URL ftp://xsun.sdct.itl.nist.gov/regrep/query/FindRNPropertyUUIDs.xml will return an AdhocQueryResponse with RegistryEntry instances for the RosettaNet properties, each embedded

with a composed ExternalIdentifier element to determine the mapping between each property external identifier XXXXXX and the UUID assigned to it by the test Registry.

5. Allow user “SOPprimary” to register a very simple DTD or Schema for a Collaboration Protocol Profile (CPP). Maybe just register the simplest version of the UN/CEFACT schema for CPP.
6. Allow any of the Organization instances to register a CPP to describe their organization. Some submissions will contain an Association instance to the registered definition for CPP and some will not. There is no Registry requirement that associationType be from the pre-defined list of association types, so some of these associations will use the association type *ValidatesTo* and some will not. There is no Registry requirement to validate any submission, and in particular there is no Registry requirement to treat CPP objectTypes in any special way.
7. Allow any of the Organization instances to register electronic components as extrinsic objects of objectType “Electronic Component”. Each may be classified according to none, one, or both of the UNSPSC or RNTDCS classification schemes. We will need at least a dozen, and probably more, registered electronic components, each with a variety of values for both required and optional attributes. These submissions will be used for subsequent testing in the maintenance and query sections below. Be sure to include registrations of repository items that supersede or replace other repository items, but leave options for further maintenance actions in the maintenance section below.
8. Allow user “SOFprimary” to register a collection of 20 electronic components and classify them by one or more UNSPSC classification nodes. There will be no explicit *repository item* for the ExtrinsicObject instances; instead, each instance will have an ExternalLink to a URL that describes that item. The SubmitObjectsRequest for this action can be found at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOFPcomponents.xml . Each extrinsic object will have a client-supplied UUID identifier that must be accepted by the test Registry as the permanent identifier for that object.
9. Allow user “SOFsecondary” to register a collection of 5 electronic components and classify them by one or more UNSPSC classification nodes. There will be no explicit *repository item* for the ExtrinsicObject instances; instead, each instance will have an ExternalLink to a URL that describes that item. The SubmitObjectsRequest for this action can be found at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOFScomponents.xml . Each extrinsic object will have a client-supplied UUID identifier that must be accepted by the test Registry as the permanent identifier for that object.
10. Allow user “SOPprimary” to register a collection of 18 electronic components and classify them by one or more UNSPSC classification nodes. There will be no explicit *repository item* for the ExtrinsicObject instances; instead, each instance will have an ExternalLink to a URL that describes that item. The SubmitObjectsRequest for this action can be found at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOPcomponents.xml . Each extrinsic object will have a client-supplied UUID identifier that must be accepted by the test Registry as the permanent identifier for that object.
11. Allow user “SOCprimary” to register a collection of 9 electronic components and classify them by one or more UNSPSC classification nodes. There will be no explicit *repository item* for the ExtrinsicObject instances; instead, each instance will have an ExternalLink to a URL that describes that item. The SubmitObjectsRequest for this action can be found at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOCcomponents.xml . Each extrinsic object will have a client-supplied UUID identifier that must be accepted by the test Registry as the permanent identifier for that object.
12. Allow user “SOPprimary” to register a second CPP profile for itself. There is no restriction on the number of CPP’s for any organization.
13. etc. (test some additional special characteristics of extrinsic objects – e.g. duplicates! Can two ExtrinsicObject instances exist with identical attributes?) Other duplicate testing will be done in the Testing Duplicates section below.

3.4 Classifications

According to the RIM specification, any RegistryObject instance may be classified by nodes or node representations from any registered classification scheme. Test requirements below will include the ability to classify any of the registered extrinsic objects by internal or external classification schemes. There is no requirement that the object being classified be owned by the submitting organization of the proposed classification. Test requirements will also include Classification submissions to classify arbitrary RegistryObject instances, including organizations, CPP’s, DTD’s, or even existing classification nodes, associations, or external identifiers, whether owned by the classification submittor or not.

The meaning of the "nodeRepresentation" attribute of Classification, RIM Section 10.3.5, is not sufficiently precise to ensure interoperability among conforming Registry implementations. The text of that section, RIM lines 1386-1391, is as follows:

"If the Classification instance represents an external classification, then the nodeRepresentation attribute is required. It is a representation of a taxonomy element from a classification scheme. It is the responsibility of the registry to distinguish between different types of nodeRepresentation, like between the classification scheme node code and the classification scheme node canonical path. This allows client to transparently use different syntaxes for nodeRepresentation."

A "best practices" or "implementor's agreement" is necessary in order to ensure that "nodeRepresentation" is handled in the same manner by conforming implementations. Since every Classification instance is dependent upon some ClassificationScheme instance, one can use information in that ClassificationScheme instance to guarantee a proper interpretation of values in the "nodeRepresentation" attribute. The "nodeType" attribute is required in every ClassificationScheme instance and in Section 10.1.3 RIM defines the meaning of each of the following values: UniqueCode, EmbeddedPath, NonUniqueCode. The OASIS Registry TC is aware that nodeRepresentation is underspecified and is considering rules similar to the following as a solution:

1. If a Classification instance has a non-null "nodeRepresentation" attribute, then it must also have a non-null "classificationScheme" attribute that references a ClassificationScheme instance.

2. If the referenced ClassificationScheme instance has "nodeType" equal to "UniqueCode" or "EmbeddedPath", then a "nodeRepresentation" value will be interpreted to be identical to the value of the "code" attribute of the intended ClassificationNode instance.

3. If the refereced Classification scheme has "nodeType" equal to "NonUniqueCode", and if a "nodeRepresentation" value begins with a solidus, i.e. "/", then the "nodeRepresentation" value will be interpreted as identical to the solidus-separated concatenation of the "code" attribute values of all classification nodes in the path that uniquely determines a node instance of that classification scheme.

4. If the refereced Classification scheme has "nodeType" equal to "NonUniqueCode", and if a "nodeRepresentation" value begins with a colon, i.e. ":", then the "nodeRepresentation" value will be interpreted as identical to the colon-separated concatenation of the "code" attribute values of all classification nodes in the path that uniquely determines a node instance of that classification scheme.

5. If the refereced Classification scheme has "nodeType" equal to "NonUniqueCode", and if a "nodeRepresentation" value begins with a reverse solidus, i.e. "\", then the "nodeRepresentation" value will be interpreted as identical to the reverse solidus-separated concatenation of the "code" attribute values of all classification nodes in the path that uniquely determines a node instance of that classification scheme.

The purpose of rules 3-5 is to allow clients multiple ways to choose a separator that doesn't conflict with characters in the supported "code" values of a classification scheme. For example, RosettaNet uses the solidus "/" as an integral part of its naming of classes in the technical dictionary, e.g. PLUGS/JACKS - COMPLEX JACK BOARDS. Others may use colons for name qualifiers on nodes, e.g. SQL:INTEGER vs Java:INTEGER.

NOTE: Later versions of RS may allow different XML representations of a nodeRepresentation, but in RS v2.0 it is simply an attribute having a string value of type LongName. Thus more sophisticated solutions to multiple code node representations will require extensions to RS.

One major advantage of the interpretation rules proposed above is that they don't require any changes to the XML elements defined in RS. Another advantage is that they could apply whether the referenced ClassificationScheme instance is *internal* or *external* as defined in RIM Section 10.1. And they offer support to a paragraph in the RIM Section 10.3, Classification, lines 1359-1362, that states:

"The attributes and methods for the Classification class are intended to allow for representation of both internal and external classifications in order to minimize the need for a submission or a query to distinguish between internal and external classifications."

In Section 10.2.5, Canonical Path Syntax, RIM defines a canonical representation of a node using solidus as a separator. No matter which character is used as a separator in a “nodeRepresentation” a combination of the “classificationScheme” and “nodeRepresentation” attributes of a Classification instance, with appropriate substitutions for the separator, could produce a string equivalent to the canonical path that would get returned by the getPath() method applied to a ClassificationNode instance, thereby ensuring that a Classification instance can always be used to identify exactly one ClassificationNode instance of an internal classification scheme.

These rules would allow client systems to use "nodeRepresentation" in a uniform way without having to know if the classification scheme is *internal* or *external* to the Registry they are connected to. Thus client software could classify submitted objects by UNSPSC classifications without having to first query the Registry to determine if UNSPSC was internal or external, and if internal, then querying the Registry to determine the UUID's for the nodes it intends to reference.

The Classification instances included with SubmitObjectsRequest elements in the test requirements of this document, may use explicit UUID’s to reference existing nodes of internal classification schemes, or they may use rules 1, 2, and 4 above to identify a classification node without use of its explicit UUID.

These test requirements exercise the above agreements by classifying a small selection of UNSPSC nodes with a corresponding RosettaNet Technical Dictionary classification. Each UN code in the following table will be classified by one or more (cf 302120) RosettaNet classification scheme nodes. The intent is that any electronic compinents registered and classified by one of these UN product codes will then be subject to the RosettaNet properties associated with the corresponding RosettaNet classification nodes. Note: Apologies to both UNSPSC and RosettaNet for obvious inconsistencies in the classifications, but they give more interesting test cases!

UNcode	UNname	RNlevel1	RNlevel2	RNlevel3
12141741	Tantalum (Ta)	CAPACITOR	FIXED	TANTAL
26121607	Fiber optic cable	FIBER-OPTIC	CABLES	
302115	Power conversion	TRANSFORMER		
30211501	Transformers	TRANSFORMER	POWER	
30211502	Power supply units	POWER SUPPLIES		
30211506	Signal converters	TRANSFORMER	SIGNAL	
30211507	Transducers	TRANSDUCERS		
30211509	Magnetic coils	COIL		
30211916	General purpose relays	RELAYS		
302120	Circuit breakers and fuses	FUSE		
302120	Circuit breakers and fuses	CURRENT CUTOFF FUSE		
30212003	Miniature circuit breakers	FUSE	IC LINKS	
30212004	Time delay fuses	FUSE	IC LINKS	
30212005	Plug fuses	FUSE	IC LINKS	
30212009	Midget fuses	FUSE	IC LINKS	
3210	Printed circuits and integrated circuits and microassemblies	CIRCUIT MODULE		
321016	Monolithic memory or smart cards	MEMORY CARDS		
32111501	Microwave diodes	MICROWAVE		
32111908	Radio frequency (RF) filters	FILTER	SIGNAL LINE	QUARTZ CRYSTAL
321215	Capacitors	CAPACITOR		
32121501	Fixed capacitors	CAPACITOR	FIXED	
32121502	Variable capacitors or varactors	CAPACITOR	VARIABLE	
32121503	Adjustable pre set capacitors	FILTER	EMI/EMC	THREE TERMINAL CAPACITORS
32121504	Capacitor networks	FILTER	EMI/EMC	NETWORK CIRCUITS
321216	Resistors	RESISTOR		
32121602	Fusistors	RESISTOR	FIXED	FUSING
32121603	Variable resistors or varistors	VARISTOR (MOV)		
411119	Indicating and recording instruments	RECORD		
41111909	Magnetic tape recorders	RECORD	MAGNETIC HEAD	
431716	Antennas	ANTENNA		
43171608	Microwave antennas	MICROWAVE		
43172204	Keyboards or keypads	KEYBOARD	FOR ECR/POS	

43172211	Voice microphones for computers	MICROPHONE UNIT		
431729	Network switches	SWITCH		
43172901	Local area network (LAN) switches	SWITCH	MECHANICAL	SIGNAL SELECTOR
43172902	Asynchronous transfer mode (ATM) switches	SWITCH	MECHANICAL	SIGNAL SELECTOR
43172903	Fiber distributed data interface (FDDI) switches	SWITCH	MECHANICAL	SIGNAL SELECTOR
43172904	Wide area network (WAN) switches	SWITCH	MECHANICAL	SIGNAL SELECTOR
46171607	Buzzers	BUZZER		
52161502	Cassette players or recorders	RECORD	COMPACT CASSETTE MECHANISMS	
52161504	Video cassette recorders	RECORD	DVD MECHANISMS	

In order to classify the UNSPSC nodes, it will be necessary to determine the UUID's assigned to those nodes by the test Registry. This can be achieved by an AdhocQueryRequest that returns the leaf ClassificationNode elements in the AdhocQueryResponse because those elements will carry the UUID as the id attribute and the UNSPSC code as the code attribute. In subsequent test cases we use UNxxxxx128bitUUID as a placeholder for the actual UUID, where xxxxx is the UNSPSC 2 to 8 digit code for a classification node.

Test Requirements

1. Allow user "SOFprimary" to determine the UUID of the UNSPSC classification scheme and the UUID's assigned by the test Registry to the 40 distinct UNSPSC nodes in the above table. Let UNSPSCCS128bitUUID represent the UUID for the UNSPSC classification scheme. After substitution for this single UUID, the AdhocQueryRequest at URL ftp://xsun.sdct.itl.nist.gov/regrep/query/FindUNSPSCnodeUUIDs.xml will return 40 ClassificationNode elements with the id and code attributes giving the UUID to UNSPSC code relationship. NOTE: the method getClassificationScheme() on ClassificationNode is invoked by a leftPredicate equal to classificationScheme, as specified by lines 1203 to 1208 in RS Section 8.2.
2. Allow user "SOFprimary" to classify the above 40 UNSPSC nodes by their corresponding RosettaNet classification nodes. The SubmitObjectsRequest to achieve this result can be found at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitUNSPSCnodeClassif.xml . The string RNTDCS128bitUUID must be replaced by the UUID of the RosettaNet Classification Scheme in the test Registry and each UNxxxxx128bitUUID placeholder must be replaced by its actual UUID before execution.
3. Allow user "SOFprimary" to classify any of its submitted extrinsic objects by nodes of any existing classification scheme, including registered classification schemes that are external.
4. Allow user "SOFprimary" to classify extrinsic objects owned by other submitting organizations.
5. Allow user "SOFprimary" to classify any of the RosettaNet Technical Dictionary classification nodes with a corresponding UNSPSC classification node.
6. Allow user "SOFprimary" to classify any of its ExternalIdentifier property code instances with a UNSPSC classification node.
7. Allow user "SOFprimary" to classify any Association instances.
8. Allow user "SOFprimary" to classify any of the Organization instances by UNSPSC.
9. Allow user "SOFprimary" to classify any of the Classification instances.
10. Allow user "SOFprimary" to classify any of the ClassificationScheme instances.

3.5 Registry packages

A registry package represents a collection of RegistryObject instances. RIM Figure 1 requires that every registry package be associated with 1 or more member objects, but the TC responsible for the specification later identified this as an error and asserted that empty packages are intended to be legal. These tests will assume that this RIM 2.0 requirement was not intended and that empty packages are legal.

According to RIM Section 9.9.2, the membership of a package is determined by Association instances with an "associationType" of *HasMember*. Registry objects may be members in any number of different packages. Packages may be created in the same request as their member instances or packages may be created empty (now valid!) and then populated by submitting associations to identify the membership. Since membership in a package is represented by associations, that membership may be subject to the confirmation requirement for associations defined in RIM Sections 9.6 through 9.8.

In Section 3.3 above, 746 RosettaNet properties were registered in this test registry and each property was assigned a UUID. We let XXXXXX_128bitUUID be an identifier for the UUID assigned to the RosettaNet property with property code XXXXXX. A substitution of the assigned UUID for this identifier will have to be made in the following test requirements before actual execution of the test cases.

The RosettaNet Technical Dictionary defines a number of named collections of properties related to electronic components. Each collection has a property set code identifier analogous to property code identifiers for individual properties, and the following tests submit that code as an external identifier. These tests will register 167 collections as packages, but in some cases the packages will be created empty and populated separately, and in some cases the packages will only be partially populated at package creation with the remaining members added as a separate step. In addition, each property set is classified by one or more RosettaNet classifications. The meaning of a property set classification is that every electronic component item classified by a node of the classification scheme may have values for each of the properties in a property set classified by that same node, and by the properties in every property set classified by a parent node of that node.

Test Requirements

1. Allow user "SOFprimary" to determine the UUID's for all 746 properties registered in Section 3.3. This can be achieved with a FilterQuery that returns all ExtrinsicObject instances having objectType = "RosettaNet Property". The query ResponseOption could be set in a way to force return of the external identifier along with the UUID's of the extrinsic objects. The Registry TC has also noted that ExternalIdentifier should have an optional attribute, registryObject, to identify the object identified. Then a FilterQuery on ExternalIdentifier *might* return both the UUID of the extrinsic object and the external identifier value. I say *might* because the attribute is NOT required. We'd need a rule to require an implementation to always return that attribute in the response to a query.
2. Suppose "SOFprimary" has already determined the UUID's for all 746 properties registered in Section 3.3. Assume that the UUID's for the External Identifier and RosettaNet Technical dictionary classification schemes are identified by EXTIDCS128bitUUID and RNTDCS128bitUUID, respectively. Then allow user "SOFprimary" to submit 167 package definitions as a single SubmitObjectsRequest. Each RegistryPackage instance will identify the package by name, will contain its RosettaNet property set code as an ExternalIdentifier instance, may be classified by one or more nodes of the RNTD classification scheme, and may contain a collection of HasMember Association instances to identify the specific properties in that collection. Most of the packages will be created with Static stability, but the following will be created with DynamicCompatible stability so they can be modified in subsequent tests: RNS-XJA292, RNS-XJA295, RNS-XJA448, and RNS-XJA541. Packages RNS-XJA292 and RNS-XJA295 are created empty and packages RNS-XJA448 and RNS-XJA541 are created with some members missing. The SubmitObjectsRequest is at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRNPropSets.xml. The actual UUID's will have to be substituted for the UUID identifiers.
3. Allow user "SOFprimary" to determine the UUID's for the empty property sets RNS-XJA448 and RNS-XJA541 registered in the previous step, and for the properties XJE209, XJF252, XJE219, XJE353, and XJF392, all registered in Section 3.3. In each case the UUID is identified by attaching "_128bitUUID" to the RosettaNet identifier. Then allow user "SOFprimary" to populate both packages with their property members. See URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRNPropSetsForXJA448andXJA541.xml for the SubmitObjectsRequest. The actual UUID's will have to be substituted for the UUID identifiers.
4. Allow user "SOFprimary" to determine the UUID's for the partially completed property sets RNS-XJA292 and RNS-XJA295, and for the properties XJF132, XJF133, XJF119, and XJF147, all registered in Section 3.3. In each case the UUID is identified by attaching "_128bitUUID" to the RosettaNet identifier. Then allow user "SOFprimary" to populate both packages with their property members. See the SubmitObjectsRequest at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitRNPropSetsMoreForXJA292andXJA295.xml. The actual UUID's will have to be substituted for the UUID identifiers.
5. Allow user "SOFsecondary" to create a package with member objects from 4 or 5 different classes, with all member objects owned by SubmitOrgFrequent.
6. Allow user "SOFprimary" to propose a package involving members owned by other organizations. Do not attempt to confirm any such membership until the Maintenance section below.

3.6 Associations

According to the RIM specification, any RegistryObject instance may be associated with any other RegistryObject instance. The associationType of any association may or may not be one of the pre-defined types defined in RIM Section 9.9.2.1. If the associationType is a RIM pre-defined type, then there may be some constraints on the objectType of the object referenced by the UUID of the sourceObject or targetObject attributes. RIM Sections 9.4 and

9.5 define *intramural* and *extramural* associations and require that extramural associations be confirmed by the owner of any source or target object of the association not owned by the submitter of the association.

A number of association instances have already been submitted to the test Registry as the result of previous actions. Every explicitly submitted RegistryObject instance results in an implicit association instance with a sourceObject reference to RegistryObject, a targetObject reference to Organization, and an associationType of *SubmitterOf*. Is a confirmation necessary? In Section 3.1, the RIM Data Type classification scheme was registered as an external scheme with an ExternalLink instance to a graphic that shows the data types. That link was accompanied by an explicit association instance with a sourceObject reference to the ExternalLink instance, a targetObject reference to the ClassificationScheme instance, and an associationType of *ExternallyLinks*. In Section 3.3, a number of electronic components were registered with ExternalLink instances to related data. Each such link was submitted along with an explicit association instance with an associationType of *ExternallyLinks*. In Section 3.5, packages of RosettaNet properties were defined by association instances with a sourceObject reference to the RegistryPackage class, a targetObject reference to the ExtrinsicObject class, and an associationType of *HasMember*. No confirmation is needed on any of these associations because the source and target objects were all owned by the same organization.

In Section 3.3 above, several users submitted ExtrinsicObject instances to describe electronic components. Each component was classified by one or more UNSPSC classification nodes. In Section 3.4 above, a number of UNSPSC classification nodes were classified by their corresponding RosettaNet Technical Dictionary classification nodes. In Section 3.5 above, collections of RosettaNet properties were classified by nodes of the Technical Dictionary classification scheme to which those properties were applicable. These definitions result in an implicit mapping from each registered electronic component to RosettaNet properties applicable to that component. In the following test requirements, some of these users will specify associations to link these electronic components to applicable RosettaNet properties. The Association instances all have an associationType of "RosettaNet Property", a sourceObject that identifies the electonic component, a targetObject that identifies the property, and a Slot instance that references the property by "name" and gives a value for the property when applied to the electronic component. Note: The "name" attributes in the Association instances are provided by the user and may or may not be identical to the official "Name" attributes of the property as defined by the extrinsic object. Queries should not assume that corresponding names are identical! Instead, queries should rely only on the association between sourceObject and targetObject.

In the following test requirements, UUID's for RosettaNet properties will be indirectly referenced just as they were in Section 3.5 above. The actual UUID values will have to be substituted in the test scripts for these indirect references before execution.

Test Requirements

1. Allow user SOFsecondary to submit properties for all electronic components previously registered by SubmitOrgFrequent. This includes all electronic component extrinsic objects submitted on behalf of SubmitOrgFrequent by users SOFprimary and SOFsecondary. The SubmitObjectRequest for this action is at URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOFpropertyValues.xml. All Property UUID placeholders must be replaced by actual Property instance UUID's before execution.
2. Allow user SOPprimary to submit properties for all electronic components previously registered by SOPprimary on behalf of SubmitOrgParent. The URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOPpropertyValues.xml contains the SubmitObjectsRequest for this action. All Property UUID placeholders must be replaced by actual Property instance UUID's before execution.
3. Allow user SOCprimary to submit properties for all electronic components previously registered by SOCprimary on behalf of SubmitOrgChild. The URL ftp://xsun.sdct.itl.nist.gov/regrep/submits/SubmitSOCpropertyValues.xml contains the SubmitObjectsRequest for this action. All Property UUID placeholders must be replaced by actual Property instance UUID's before execution. NOTE: The two organizations SubmitOrgParent and SubmitOrgChild are treated separately because the Registry does not automatically grant privileges between them for managing one another's objects.
4. Allow owners of various repository items to submit new ResponsibleFor associations to name the organization responsible for that repository item. How do these get confirmed – must the ResponsibleOrg confirm each one individually?
5. Allow user "SOFprimary" to associate his organization with itself , i.e. sourceObject = targetObject, with "Identity" as the name of the association and with associationType *EquivalentTo*. This is not prohibited and should not produce an error.
6. etc.

3.7 External identifiers

An ExternalIdentifier instance provides alternate identifiers for a registry object. The required identificationScheme attribute for an external identifier points to a classification scheme known to the Registry. The intent (we believe!) is that the name attribute then references a node from this classification scheme, but that is not explicitly stated in the specification (see RIM Section 7.9.2; maybe the 2nd sentence intended to say this!). A user should be able to obtain the Registry created UUID for an object if the name of an *identification scheme*, e.g. DUNS, and a value from that scheme, e.g. DUNS number, is known. Note: in RIM, an *identification scheme* is NOT the same as a *classification scheme*. The link from an external identifier to the object being identified is not represented as an Association instance, so it is not subject to Association confirmation as discussed in RIM Section 9.6. Thus, with appropriate privileges, any submitter should be able to submit external identifiers for any RegistryObject instance, including those registry objects not owned by the submittor.

A number of external identifiers for RosettaNet leaf classifications, for RosettaNet properties, and for collections of RosettaNet properties, have already been created by test requirements in previous sections. The following test requirements address other issues related to external identifiers.

Test Requirements

1. Allow organizations to submit a DUNS number identifier for themselves.
2. Allow organizations to submit identifiers for registry objects they do NOT own. It may be necessary to establish a privilege for doing this first!
3. Attempt to submit an ExternalIdentifier instance whose value attribute duplicates a value already in the registry for that identificationScheme and name. This is not explicitly prohibited by the Registry specification, but it probably should be. E.g. Try to submit two different DUNS numbers for the same organization.
4. etc. (additional test requirements to test other issues)

3.8 External links

An ExternalLink instance represents a named URI to content not in the Registry. In these test requirements it will be used to hold web page URL's for various registry users or organizations, or to hold URL's to GIF or JPEG graphics of repository items. The externalURI attribute is the only attribute required beyond those of any registry object. However, in these test requirements, the name attribute will never be null and will always identify the link name of the URL.

RIM Figure 1 requires that every external link be associated with 1 or more registry objects. The TC responsible for Registry has since declared this to be an error in the specification and has removed that requirement. It will now be possible to have ExternalLink instances that do not link to any other RegistryObject instance.

If an ExternalLink instance is a link for a RegistryObject instance, then RIM Section 9.2.2.1 requires that this association be maintained in the Registry with the associationType *ExternallyLinks*. All such external link associations are subject to the confirmation requirements for associations defined in RIM Sections 9.6 through 9.8.

Previous submissions to this test Registry have already resulted in some ExternalLink instances. In Section 3.1, the RIM Data Type classification scheme was registered as an external scheme with an ExternalLink instance to a graphic that shows the data types. That link results in an implicit association instance with a sourceObject reference to the ExternalLink instance, a targetObject reference to the ClassificationScheme instance, and an associationType of *ExternallyLinks*. In Section 3.3, a number of electronic components were registered with ExternalLink instances to related data. Each such link produces an implicit association instance with an associationType of *ExternallyLinks*. No confirmation is needed for any of these associations because the source and target objects were all submitted and owned by the same organization.

Test Requirements

1. Allow organizations to set up external links to their organization home pages using the name of the organization as the name attribute of the link.

2. Allow owners of "Electronic Components" to submit external links to a Web Page that gives a GIF or JPEG graphic of the componment. [THIS MAY ALREADY BE DONE IN SECTION 3.3, step #6]
3. Allow arbitrary users to propose external links for objects they do NOT own. Do not confirm any such proposed associations until the Maintenance section below.
4. Allow user "SOFprimary" to set up an ExternalLink instance that is associated with itself and with no other registry object. This should be legal! Thus a user should be able to submit an external link that won't really get used until later. But, how does the user determine the UUID of it?
5. Allow an arbitrary user to set up an external link that is not associated with any other registry object (RIM Figure 1 says every external link must be associated with at least one registry object, but that has been changed by an errata approved by the Registry TC).

3.9 Slot extensions

Slot instances provide a dynamic way to extend the registry information model by adding arbitrary attributes to registry objects. Slot instances are not subtypes of RegistryObject so they do not have a required UUID and do not have any of the other attributes of a registry object. Instead, a slot instance is composed of a name, a slotType, and a collection of values. According to RIM Figure 1, slots are dependent upon the registry object they are grouped with, but unlike ExternalIdentifier instances they do NOT have a required attribute for the UUID of the object they are part of. Thus they have no separate existence. If there is more than one value for a slot, there is no requirement for the Registry to maintain a specific order of those values.

Test Requirements

1. Allow users that have submitted extrinsic objects of type "Electronic Component" to associate those objects with one or more RosettaNet properties, where each association will carry a "slot" value of that property for that component. The slotType attribute will be "RosettaNet Property", slot name will be RosettaNet property code or the name of the property (the UUID of the property is the targetObject of the association), and the value will carry a text literal representation of the value the sourceObject has for the property identified by the targetObject. Since a Slot can have multiple values, the Slot may carry both the RrosettaNet property code and the name of the property! The test cases will abide by the constaints on the property value, such as its data type and enumeration constraints, but the Registry specification has no way to enforce these constraints in a standard manner.
2. Allow user "SOFprimary" to submit a Slot with multiple values. [Already DONE in step #1 above and in SubmitSOF]
3. Etc. (additional test requirements to test other Slot issues)

3.10 Services

Read all of the RIM sections dealing with Service, ServiceBinding, and SpecificationLink. Service is a subtype of RegistryEntry so has all of the attributes of a RegistryEntry instance. Each service maps to a collection of zero or more ServiceBinding instances, but this mapping is NOT represented as an Association instance so is not subject to association confirmation requirements defined in RIM Sections 9.6 through 9.8. Each service binding maps to a collection of one or more SpecificationLink instances, but this mapping is NOT represented as an Association instance, etc.

Test Requirements

[TO BE COMPLETED]

4. Maintenance testing

The purpose of this step is to exercise registry services for approving, confirming, updating, deprecating, replacing, deleting, or superseding existing objects. Subsections 4.1 and 4.2 on Approval and Confirmation shall be executed first. If possible, tests in all of the other subsections will leave the Registry in a state compatible with all of the other tests those subsections so that with the exception of approvals and confirmations, the section as a whole will be independent of the order of execution of the individual test cases.

The following subsections identify high-level requirements for each category of maintenance.

4.1 Approval of RegistryEntry instances

RIM Section 7.5.6.1 specifies that when a RegistryEntry instance is first submitted it will have a status value of *Submitted* and subsequently it must be changed to *Approved* before it can be seen or used by other business users. In Section 7.7 RS provides an Approve Objects Protocol. As discussed above, RIM and RS allow different interpretations of when and how this approval process takes place. The intent in this section is for the RA or the owner of a submitted object to approve some RegistryEntry instances that have not yet been approved and that we want visible to other users in subsequent test requirements. Some RegistryEntry instances will be left unapproved for subsequent privilege testing.

Test Requirements

1. Approve some of the existing non-approved RegistryEntry instances.
2. Leave some RegistryEntry instances non-approved for subsequent test requirements.
3. Etc. (test requirements for other approval issues)

4.2 Confirmation of Association instances

RIM Sections 9.4 and 9.5 define *intramural* and *extramural* associations and require that extramural associations be confirmed by the owner of any source or target object of the association not owned by the submitter of the association. The intent in this section is for the affected owners of sourceObject or targetObject references in mutli-party Association instances to confirm those associations they want visible to other business users.

The RIM specification is not clear about whether or not an affected owner can see the Association if it was submitted by another owner. It may be necessary for a second affected owner to know in advance what is to be confirmed. It may not be possible to query associations to see which ones need confirmation by your organization. However, these test requirements will assume that appropriate privileges have been granted so that any user of any affected organization is able to see un-confirmed associations in which that organization is the owner of an object referenced by a sourceObject or targetObject value.

Test Requirements

1. Confirm some of the existing non-confirmed associations.
2. Leave some associations non-confirmed for subsequent test requirements.
3. Etc. (test requirements for other confirmation issues)

4.3 Updateable attributes

RIM defines each attribute of each class to be mutable or not mutable. If a UUID attribute is mutable it means ???. If a non-UUID attribute is mutable it means that the value of that attribute is updateable by a user. If a non-UUID attribute is not mutable it means that the value of that attribute is not under update control of an end user; instead, the value can only be set, updated, or replaced by the Registration Authority.

In Section 7.4, RS provides an Update Objects Protocol for updating the status attribute of a RegistryEntry instance. The following requirements test various types of updates.

Test Requirements

1. Have the owner of some objects update some mutable non-UUIDattributes of those objects.
2. Have the owner of some objects update some mutable UUID attributes of those objects.
3. Allow the owner of a classification scheme that has Dynamic stability to update the parent attribute of some nodes to re-structure the classification hierarchy. See if queries return correct results both before and after the re-structuring. E.g. Use RosettaNet Property classifications by Integer as defined, and then with Integer re-structured as a separate level 2 node under Number.
4. Etc.

4.4 Deprecation and other change of status

Section 7.5.6 of RIM specifies pre-defined status values for the status attribute of any RegistryEntry instance. These include *Submitted*, *Approved*, *Deprecated*, and *Withdrawn*. In Section 7.7 RS provides an Approve Objects Protocol that allows a "client" to approve one or more previously submitted repository items, but it doesn't say if the client is the RA, the owner, or some other user. RIM is silent on which users can "see" unapproved items but RS Section 7.7, line 987, implies that items must be approved before they will become available for use by business parties. These test requirements assume that ultimate approval authority resides with the RA, but that the RA has granted the approval privilege to the owner of each object. However, test requirements will assume that owners of un-approved registry entries can set up associations of those entries with other registry objects before approval; otherwise, it would not be possible to submit a collection of related objects and see what the effect is before final approval.

Test Requirements

1. Allow user "SOFsecondary" to determine the UUID's of a collection of RosettaNet classification nodes that have been declared as "OBSOLETED" by RosettaNet. When RNTDCS nodes were added to the Registry in Section 3.2, only the non-OBSOLETED leaf nodes were assigned a RosettaNet ExternalIdentifier. Thus the nodes to be deprecated must be identified by their RosettaNet classification node name rather than by their external identifiers. The list of node names is available at URL ftp:??? etc, and the Query to retrieve the UUID,s of that list of names is at URL ftp:??? (see file OBSOLETEDconnectors).
2. Allow user "SOFsecondary" to update a group of nodes of the RosettaNet classification scheme, to Deprecate those nodes that have been declared "OBSOLETED" by RosettaNet.
3. Etc.

4.5 Replacement

What do RIM and RS say about replacing an object with something else. Are the rules different for RegistryEntry instances versus Classification and Association instances?

Test Requirements

1. Allow
2. Etc.

5. Error Handling and Warnings

The Registry will return an error to the client if an attempt is made to submit an object that violates an integrity constraint or duplicates an existing instance. The following sections discuss requirements for the Registry to catch certain types of client errors: etc.

5.1 RS Section 7.3.5, SubmitObjects

In Section 7.3.5, the RS specification defines the following Error Handling rules for SubmitObjects:

Referenced object not found! Error!

Associations not allowed to connect to deprecated objects! Error!

Object status set by Registration Authority! Warning

Object version set by Registration Authority! Warning

Test Requirements

1. Attempt to submit an Association instance with an associationType of *HasMember* and with a sourceObject that is NOT a RegistryPackage UUID. And with a targetObject that is NOT a RegistryEntry UUID (mistake in line 1172??)
2. Attempt to submit an Association instance with an associationType of *ExternallyLinks* and with a sourceObject that is NOT an ExternalLink UUID. And with a targetObject that is NOT a RegistryEntry UUID (mistake in line 1172??)
3. Attempt to submit an Association instance with an associationType of *HasMember* and with a sourceObject of objectType RegistryPackage that is equal to the targetObject, i.e. a statement that a package is a member of itself! Although not explicitly prohibited by the RIM or RS specifications, this would violate a mathematical assumption that sets are never members of themselves. This situation merits at least a Warning if not an Error!
4. Have user "SOFprimary" attempt to submit an Association instance with an associationType of *SubmitterOf* and with the sourceObject the UUID to SubmitOrgFrequent" and with targetObject the UUID of an object previously submitted by SubmitOrgFrequent. Should return an error since only the Registration Authority should be able to use this associationType.
5. Attempt to submit an Association instance with an associationType of *SubmitterOf* and with a sourceObject that is NOT an Organization UUID.
6. Attempt to submit an Association instance with an associationType of *ResponsibleFor* and with a sourceObject that is NOT an Organization UUID.
7. Attempt to submit an ExternalIdentifier instance whose value attribute duplicates a value already in the registry for that identificationScheme and name.

5.2 RS Section 7.4.3, UpdateObjects

In Section 7.4.3, the RS specification defines the following Error Handling rules for UpdateObjects:

Object not found! Error

Not authorized! Error

Referenced object not found! Error

Associations not allowed to connect to deprecated objects! Error!

Object status set by Registration Authority! Warning

Object version set by Registration Authority! Warning

Registry entries with stable status should not be updated! Warning!! (Why NOT Error?)

Test Requirements

1. Attempt to have a user who is not an owner of an object update a mutable attribute of that object.
2. Attempt to have the owner of an ExtrinsicObject instance update its status attribute.
3. Attempt to have the owner of an ExtrinsicObject instance update its majorVersion attribute.
4. Attempt to have the owner of an ExtrinsicObject instance update its minorVersion attribute.
5. Attempt to have the owner of an ExtrinsicObject instance update its expiration attribute. Determine how the Registration Authority can preclude owners from updating the expiration date to something longer than a set time span, e.g. 5 years!
6. Attempt to have the owner of an ExtrinsicObject instance whose status is Static, update its status attribute to something other than Static.
7. Attempt to have the owner of an ExtrinsicObject instance whose status is DynamicCompatible, update its status attribute to Dynamic.

5.3 RS Section 7.7.3, ApproveObjects

In Section 7.7.3, the RS specification defines the following Error Handling rules for ApproveObjects:

Object not found! Error

Not authorized! Error

Only RegistryEntry instances may be approved! Error

Object status is already approved! Warning

Test Requirements

1. Attempt …
2. Etc.

5.4 RS Section 7.8.3, DeprecateObjects

In Section 7.8.3, the RS specification defines the following Error Handling rules for DeprecateObjects:

Object not found! Error

Not authorized! Error

Only RegistryEntry instances may be deprecated! Error

Object status is already deprecated! Warning

Test Requirements

1. Attempt …
2. Etc.

5.5 RS Section 7.9.3, RemoveObjects

In Section 7.9.3, the RS specification defines the following Error Handling rules for RemoveObjects:

Object not found! Error

Not authorized! Error

Test Requirements

1. Attempt …
2. Etc.

6. Query testing

The purpose of this step is to identify a set of XML requests to execute queries against administrative structures, e.g. audit trails, and to execute some of the more sophisticated query requirements specified by RS. Some requests will be executed against the Registry instances created by the preceding population and maintenance steps, but others may require additional LifeCycle requests to modify instances or add new instances to put the Registry into the proper state for testing a specific query. Each test will leave the Registry in the same state as it was before the initial request of that test so that this section of the test suite will be independent of the order of execution of the individual test cases.

A Registry AdhocQueryRequest may be a FilterQuery instance or an SQLQuery instance. These test requirements only address FilterQuery instances because they are the only query types required of every conforming Registry implementation. Each AdhocQueryRequest results in the return of an AdhocQueryResponse which is of type FilterQueryResult or SQLQueryResult. An AdhocQueryRequest contains a ResponseOption element with a returnType attribute that identifies one of the following return options:

- ObjectRef – to return just ObjectRef instances with the UUID's of the identified objects,
- RegistryObject – to return just RegistryObject instances with associated attributes of the identified objects,
- RegistryEntry – to return just RegistryEntry instances with associated attributes of the identified objects,
- LeafClass – to return XML elements that correspond to leaf objects defined by RIM Figure 2,
- LeafClassWith RepositoryItem – to return ExtrinsicObject elements along with their associated repository items.

In addition, each ResponseOption element has a Boolean attribute "returnComposedObjects" to indicate whether or not the whole hierarchy of composed objects are returned with the registry objects.

Because of the way RIM elements are defined by RS, there may be alternative ways to return the same information. For example, since Classification and ExternalIdentifier elements may be contained in a parent RegistryObject element, or may be stand-alone elements in a LeafRegistryObjectList, an implementation may return results that validate to different element structures. IS THIS GOOD OR BAD? Should there be a more refined interpretation of the choices defined above to avoid this anomaly? Or should there be an additional attribute on ResponseOption for the client to declare if elements should be retuned in "nested" format or "flat" format?

6.1 AuditableEvent queries

The audit trail instances are created and maintained entirely by the Registration Authority. The following test assertions address whether the Registry kept a proper audit trail of all of the preceding population and maintenance actions.

6.2 RegistryObject queries

There are several different varieties of RegistryObject queries. The following test assertions cover at least one situation for each distinct variety.

6.3 RegistryEntry queries

[TO BE DEVELOPED]

6.4 Association queries

[TO BE DEVELOPED]

6.5 Organization queries

[TO BE DEVELOPED]

6.6 Service queries

[TO BE DEVELOPED]

7. Conclusion

Talk about how the more detailed test assertions and executable test cases will get developed.
[TO BE DEVELOPED]