OASIS ebXML RegRep Service and Protocols (ebRS) Version 4.0

Draft

October 1, 2008

Specification URIs:

This Version:

http://docs.oasis-open.org/regrep/4.0-draft-1/specs/core/regrep-rs-4.0-draft-1.html

http://docs.oasis-open.org/regrep/4.0-draft-1/specs/core/regrep-rs-4.0-draft-1.odt

http://docs.oasis-open.org/regrep/4.0-draft-1/specs/core/regrep-rs-4.0-draft-1.pdf

Previous Version:

http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rs-3.0-os.html

http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rs-3.0-os.odt

http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rs-3.0-os.pdf

Latest Version:

http://docs.oasis-open.org/regrep/4.0-draft-1/specs/core/regrep-rs-4.0-draft-1.html

http://docs.oasis-open.org/regrep/4.0-draft-1/specs/core/regrep-rs-4.0-draft-1.odt

http://docs.oasis-open.org/regrep/4.0-draft-1/specs/core/regrep-rs-4.0-draft-1.pdf

Latest Approved Version:

http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rs-3.0-os.html

http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rs-3.0-os.odt

http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rs-3.0-os.pdf

Technical Committee:

OASIS ebXML Registry TC

Chair(s):

Kathryn Breininger, Boeing

Editor(s):

Farrukh Najmi, Wellfleet Software

Nikola Stojanovic, RosettaNet

Contributors:

Kathryn Breininger, Boeing
Carl Mattocks, MetLife
Farrukh Najmi, Wellfleet Software
Oliver Newell, MIT Lincoln Labs
Nikola Stojanovic, RosettaNet
David Webber, Individual

Related Work:

This specification replaces or supercedes:

- [specifications replaced by this standard - OASIS as well as other standards organizations]
- [specifications replaced by this standard - OASIS as well as other standards organizations]

This specification is related to:

- [specifications related to this standard - OASIS as well as other standards organizations]
- [specifications related to this standard - OASIS as well as other standards organizations]

Declared XML Namespace(s):

This following table lists the namespace prefixes defined and / or referenced by this specification.

Namespace Prefix	Namespace URI	Defining Specification
enc	http://www.w3.org/2003/05/soap-encoding	A normative XML Schema [XML Schema Part 1], [XML Schema Part 2] document for the "http://www.w3.org/2003/05/soap-encoding" namespace can be found at http://www.w3.org/2003/05/soap-encoding.
env	http://www.w3.org/2003/05/soap-envelope	SOAP Version 1.2 Part 1. A normative XML Schema [XML Schema Part 1], [XML Schema Part 2] document for the "http://www.w3.org/2003/05/soap-envelope" namespace can be found at http://www.w3.org/2003/05/soap-envelope.
lcm	urn:oasis:names:tc:ebxml-regrep:xsd:lcm:4.0	ebXML RegRep Services and Protocols 4.0 (ebRS)
mime	http://schemas.xmlsoap.org/wsdl/mime/	WSDL namespace for WSDL MIME binding.
query	urn:oasis:names:tc:ebxml-regrep:xsd:query:4.0	ebXML RegRep Services and Protocols 4.0 (ebRS)
rim	urn:oasis:names:tc:ebxml-regrep:xsd:rim:4.0	ebXML RegRep Registry Information Model 4.0 (ebRIM)
rs	urn:oasis:names:tc:ebxml-regrep:xsd:rs:4.0	ebXML RegRep Services and Protocols 4.0 (ebRS)
wsdl	http://schemas.xmlsoap.org/wsdl/	WSDL 1.1 namespace defined by WSDL 1.1 specification.
xs	http://www.w3.org/2001/XMLSchema	XML Schema [XML Schema Part 1], [XML Schema Part 2] specification
xsi	"http://www.w3.org/2001/XMLSchema-instance	W3C XML Schema specification [XML Schema Part 1], [XML Schema Part 2].

Table 1: Namespaces Used

Abstract:

This document defines the services and protocols for an ebXML Registry

A separate document, ebXML Registry: Information Model [ebRIM], defines the types of metadata and content that can be stored in an ebXML Registry.

Status:

This document is a draft specification for review, revision and approval by the OASIS ebXML Registry TC.

Technical Committee members should send comments on this specification to the Technical Committee's email list. Others should send comments to the Technical Committee by using the

"Send A Comment" button on the Technical Committee's web page at http://www.oasis-open.org/committeees/regrep/.

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/regrep/ipr.php.

The non-normative errata page for this specification is located at http://docs.oasis-open.org/regrep/4.0-draft-1/specs/core/errata.pdf

Notices

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS' procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", [insert specific trademarked names, abbreviations, etc. here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications,

while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.

Table of Contents

Illustration Index

Index of Tables

1 Introduction

[All text is normative unless otherwise indicated.]

1.1 Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119 .

1.2 Normative References

[RFC 2119] S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels*. IETF RFC 2119, March 1997. http://www.ietf.org/rfc/rfc2119.txt.

[Reference] [reference citation]

1.3 Non-normative References

[Reference] [reference citation]

[Reference] [reference citation]

2 Overview

ebXML RegRep is a standard defining the service interfaces, protocols and information model for an integrated registry and repository. The repository is stores digital content while the repository stores metadata that describes the content in the repository.

A more detailed overview of this specification is provided in the ebXML RegRep 4.0 Primer [Primer].

2.1 The Library Analogy

To explain what is an ebXML RegRep we use the following familiar analogy. The ebXML Registry-Repository is to digital content, what your local library is to books and other published content. We make this analogy clearer with the comparisons made in the following table:

Your Local Library	ebXML RegRep
Manages books and all types of published material	Manages all types of digital content
Has book shelves containing books and other published material	Has a "repository" containing all types of digital content
Has a card catalog that describes the published material that is available in the book shelves	Has a "registry" that describes the digital content that is available in the repository
Multiple libraries can voluntarily participate in a cooperative network and offer a unified service	Multiple ebXML Registry-Repository's can voluntarily participate in a cooperative network and offer a unified service

Table 2: ebXML RegRep comparison with your local library

2.2 Core Specifications

ebXML RegRep standard consists of two core specifications:

- The *ebXML Registry Information Model (ebRIM)* specification defines the types of metadata that can be stored in an ebXML Registry.
- The *ebXML Registry Services and Protocols (ebRS)* defines the services provided by an ebXML Registry and the protocols used by clients of the registry to interact with these services.

3 QueryManager Interface

The QueryManager interface allows a client to invoke queries on the server. A server MUST implement the QueryManager interface as an endpoint.

3.1 Parameterized Queries

A server may support any number of pre-configured queries known as *Parameterized Queries*, that may be invoked by clients. This specification defines a number of canonical queries that are standard queries that MUST be supported by a server. Implementations and deployments may define additional parameterized queries beyond the canonical queries defined by this specification.

A client invokes a parameterized query supported by the server by identifying its unique id as well as values for any parameters supported by the query.

A parameterized query is represented by a specialized RegistryObject called ParameterizedQuery object which is defined by ebRIM. The definition of a ParameterizedQuery may contain any number of Parameters supported by the query.

3.2 Adhoc Queries

A client may invoke a client specific ad hoc query using the canonical ArbitraryQuery query defined by this specification. Due to the risks associated with un-controlled ad hoc queries, a deployment MAY choose to restrict the invocation of the ArbitraryQuery to specific roles. This specification does not define a standard query expression syntax for ad hoc queries. A server MAY support any number of query expression syntaxes for ad hoc queries.

3.3 Query Protocol

A client invokes a parameterized query using the *Query* protocol supported by the executeQuery operation of the QueryManager interface.

A client initiates the Query protocol by sending an QueryRequest message to the QueryManager endpoint.

The QueryManager sends an QueryResponse back to the client as response. The QueryResponse contains a set of objects that match the query.

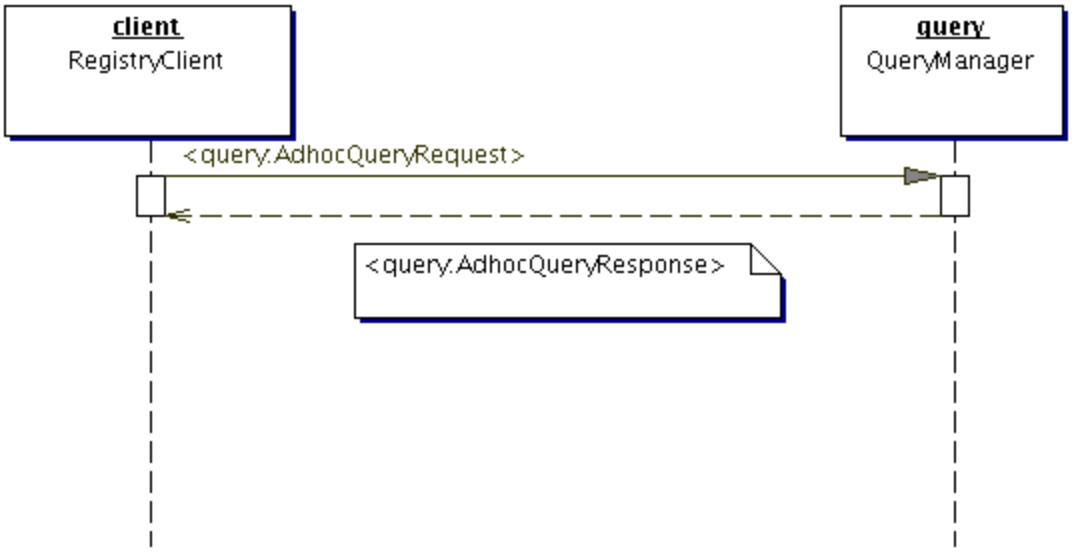

Illustration 1: Query Protocol *Need to update figure??*

3.3.1 QueryRequest

The QueryRequest message is sent by client to the QueryManager interface to invoke a query.

3.3.1.1 Syntax

```
  <element name="QueryRequest">
    <annotation>
      <documentation xml:lang="en">Request to invoke a query.</documentation>
    </annotation>
    <complexType>
      <complexContent>
        <extension base="rs:RegistryRequestType">
          <sequence>
            <element maxOccurs="1" minOccurs="1" ref="tns:ResponseOption"/>
            <element ref="rim:QueryInvocation"/>
          </sequence>
          <attribute default="application/ebrim+xml" name="format"
type="string" use="optional"/>
          <attribute name="language" type="string" use="optional"/>
          <attribute default="0" name="startIndex" type="integer"/>
          <attribute default="-1" name="maxResults" type="integer"/>
          <attribute default="0" name="depth" type="integer"/>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

3.3.1.2 Example

TODO??

3.3.1.3 Description

Issue: Need to list items in order of schema, schema should be in alpha order??

- Element ResponseOption - This required element allows the client to control the format and content of the QueryResponse generated by the registry in response to this request.
- Element QueryInvocation - This element identifies a parameterized query and supplies values for its parameters.
- Attribute maxResults - This optional attribute specifies a limit on the maximum number of results the client wishes the query to return. If unspecified, the registry SHOULD return either all the results, or in case the result set size exceeds a registry specific limit, the registry SHOULD return a sub-set of results that are within the bounds of the registry specific limit.
- Attribute startIndex - This optional integer value is used to indicate which result must be returned as the first result when iterating over a large result set. The default value is 0, which returns the result set starting with index 0 (first result).
- Attribute format - This optional attribute specifies the format of the response desired by the client. The value of this attribute MUST be a registered Internet Media Types with IANA. The default value is "application/*ebrs*+xml?? Nikola to look at RFC 3023" which returns the response in ebRS QueryResponse format.
- Attribute language - This optional attribute specifies the natural language of the response desired by the client. The default value is to return the response with all available natural languages. What coding standard to base this on. ISO 639, 639-1, 639-2??
- Attribute depth - This optional attribute specifies the pre-fetch depth of the response desired by the client. A depth of 0 (default) indicates that the server MUST return only those objects that match the query. A depth of N where N is greater that 0 indicates that the server MUST also return objects that are reachable by N levels of references via attributes that reference other objects. A depth of -1 indicates that the server MUST return all objects within the transitive closure of all references from objects that matches the query.

3.3.1.4 Response

This request returns QueryResponse as response.

3.3.1.5 Exceptions

Issue: Need to define more fine-grained exception hierarchy??

In addition to the exceptions common to all requests defined in ??, the following exceptions MAY be returned:

- InvalidQueryException: signifies that the query syntax or semantics was invalid. Client must fix the query syntax or semantic error and re-submit the query

3.3.2 Element QueryInvocation

A client specifies a <rim:QueryInvocation> element within an QueryRequest to specify the parameterized query being being invoked as well as the values for its parameters.

3.3.2.1 Syntax

```
 <complexType name="QueryInvocationType">
  <annotation>
    <documentation xml:lang="en">
    Represents invocation of a parameterized query.
    </documentation>
  </annotation>

  <!--
  References a ParameterizedQuery.
  Note that reference may be static or dynamic.
  -->
  <sequence>
    <element maxOccurs="unbounded" minOccurs="0" ref="tns:Slot"/>
  </sequence>
  <attribute name="query" type="tns:referenceURI" use="required"/>
</complexType>
<element name="QueryInvocation" type="tns:QueryInvocationType"/>
```

3.3.2.2 Description:

- *Element Slot* - Each Slot element specifies a parameter value for a parameter supported by the query. The slot name MUST match a parameterName attribute within a Parameter's definition within the ParameterizedQuery (inconistent on use of rim: prefix??) definition. The slot value provides a value for the parameter. The slot value's type MUST match the dataType attribute for the Parameter's definition within the ParameterizedQuery. Order of parameters is not significant.
- Attribute query - The value of this attribute must be a reference to a parameterized query that is supported by the server. Need to change type from referenceURI since id may not be a URI any more and also needs to support dynamic references??

3.3.3 Element ResponseOption

A client specifies a ResponseOption structure within an QueryRequest to indicate the format of the results within the corresponding QueryResponse.

3.3.3.1 Syntax

```
<complexType name="ResponseOptionType">
  <attribute default="RegistryObject" name="returnType">
    <simpleType>
      <restriction base="NCName">
        <enumeration value="ObjectRef"/>
        <enumeration value="RegistryObject"/>
        <enumeration value="LeafClass"/>
        <enumeration value="LeafClassWithRepositoryItem"/>
      </restriction>
    </simpleType>
  </attribute>
  <attribute default="false" name="returnComposedObjects" type="boolean"/>
</complexType>
<element name="ResponseOption" type="tns:ResponseOptionType"/>
```

3.3.3.2 Description:

- Attribute returnComposedObjects - This optional attribute specifies whether the RegistryObjects returned should include composed objects as defined by Figure 1 in [ebRIM]. The default is to return all composed objects.
- Attribute returnType - This optional attribute specifies the type of RegistryObject to return within the response. Values for returnType are as follows:
  - *ObjectRef* - This option specifies that the QueryResponse MUST contain a collection of <rim:ObjectRef> elements. The purpose of this option is to return references to registry objects rather than the actual objects.
  - *RegistryObject* - This option specifies that the QueryResponse MUST contain a collection of <rim:RegistryObject> elements.
  - *LeafClass* - This option specifies that the QueryResponse MUST contain a collection of elements that correspond to leaf classes as defined in [RR-RIM-XSD].
  - *LeafClassWithRepositoryItem* - This option is same as LeafClass option with the additional requirement that the response include the RepositoryItems, if any, for every <rim:ExtrinsicObject> element in the response.

  If "returnType" specified does not match a result returned by the query, then the registry *must* use the closest matching semantically valid returnType that matches the result. For example, consider a case where OrganizationQuery is asked to return LeafClassWithRepositoryItem. As this is not possible, QueryManager will assume LeafClass option instead.

3.3.4 QueryResponse

The QueryResponse message is sent by the QueryManager in response to an QueryRequest when the format requested by the client is the default ebrs format.

3.3.4.1 Syntax

```
<element name="QueryResponse">
    <annotation>
      <documentation xml:lang="en">
        The response includes a RegistryObjectList which has zero or more
        RegistryObjects that match the query specified in QueryRequest.
      </documentation>
    </annotation>
    <complexType>
      <complexContent>
        <extension base="rs:RegistryResponseType">
          <sequence>
            <element ref="rim:RegistryObjectList"/>
          </sequence>
          <attribute default="0" name="startIndex" type="integer"/>
          <attribute name="totalResultCount" type="integer" use="optional"/>
        </extension>
      </complexContent>
    </complexType>
  </element>
```


3.3.4.2 Example

TODO??

3.3.4.3 Description:

- Element RegistryObjectList - This is the element that contains the RegistryObject instances that matched the specified query.
- Attribute startIndex - This optional integer value is used to indicate the index for the first result in the result set returned by the query, within the complete result set matching the query. By default, this value is 0.
- Attribute totalResultCount - This optional parameter specifies the size of the complete result set matching the query within the registry. When this value is unspecified, the client should assume it is the size of the result set contained within the result.

3.4 Parameterized Query Definition

A parameterized query is defined by submitting a <rim:ParameterizedQuery> object to the server using the submitObjects protocol. A detailed specification of the <rim:ParameterizedQuery> object is defined in ebRIM. The definition of a parameterized query includes detailed specification of each supported parameter including its name, description, data type, cardinality and domain.

3.5 Canonical Query: ArbitraryQuery

Issue: Nikola has issue with this??

The canonical query ArbitraryQuery allows clients to invoke a client-specified ad hoc query in a client-specified query expression syntax that is supported by the server. This specification does not require a server to support a specific query expression syntax. It is likely that servers may support one or more common syntaxes such as SQL-92, XQuery, XPath, SPARQL, Search-WS, OGC Filter etc.

3.5.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
queryExpression	This parameter's value is an element of type <rim:QueryExpressionType> as defined by ebRIM. The queryExpression is used to carry the query expression for an ad hoc query. The queryExpression element also specifies the query language syntax via its queryLanguage attribute.	slot		1

3.5.2 Query Semantics

- The server MUST return a InvalidQueryException fault message if the queryLanguage used by the queryExpression is not supported by the server
- The server SHOULD return an AuthorizationException fault message if the client is not authorized to invoke this query
- The server MUST return the objects matching the query if the query is processed without any exceptions

3.6 Canonical Query: BasicQuery

The canonical query BasicQuery allows clients to query for RegistryObjects by their name, description, type, status and classifications.

3.6.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
classifications	Set whose elements are path attribute values to ClassificationNodes. Matches RegistryObjects that have a classification whose classificationNode attribute value matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value When multiple values are specifies it implies a logical AND operation.	string		0..*
description	Matches rim:RegistryObject/rim:Description/rim:LocalizedString/@value	string		0..1
matchOnAnyParameter	If true then use logical OR between predicates for each parameter	boolean	false	0..1
name	Matches rim:RegistryObject/rim:Name/rim:LocalizedString/@value	string		0..1
objectType	Matches RegistryObjects whose objectType attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value	string		0..1
status	Matches RegistryObjects whose status attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value	string		0..1

3.6.2 Query Semantics

- This query has several optional parameters
- Each parameter implies a predicate within the underlying query
- Predicates for each supplied parameter are combined using with an implicit LOGICAL AND if matchOnAnyParameter is unspecified or false. If it is specified as true then predicates for each supplied parameters are combined using a LOGICAL OR
- If an optional parameter is not supplied then its corresponding predicate MUST NOT be included in the underlying query

3.7 Canonical Query: FindByIdAndType

The canonical query FindByIdAndType allows clients to find RegistryObjects based upon their the value of their id and objectType attributes.

3.7.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
id	Matches rim:RegistryObject/@id. Allows use of "%" wildcard character to match multiple characters. Allows use of "?" wildcard character to match a single character.	string		0..1
objectType	Matches RegistryObjects whose objectType attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value	string		0..1

3.7.2 Query Semantics

- The server MUST return the objects matching the query if the query is processed without any exceptions

3.8 Canonical Query: FindAssociations

The canonical query FindAssociations query allows clients to find Associations that match the specified criteria.

3.8.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
associationType	Matches Associations where	string		0..1

	rim:/Association/@associationType references a RegistryObject whose objectType attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value			
matchOnAnyParameter	If true then use logical OR between predicates for each parameter	boolean	false	0..1
sourceObjectId	Matches rim:Association/@sourceObject. Allows use of "%" wildcard character to match multiple characters. Allows use of "?" wildcard character to match a single character.	string		0..1
sourceObjectType	Matches Associations whose sourceObject attribute references a RegistryObject whose objectType attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value	string		0..1
targetObjectId	Matches rim:Association/@targetObject. Allows use of "%" wildcard character to match multiple characters. Allows use of "?" wildcard character to match a single character.	string		0..1
targetObjectType	Matches Associations whose targetObject attribute references a RegistryObject whose objectType attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value	string		0..1

3.8.2 Query Semantics

- All parameters are optional
- The server MUST return the objects matching the query if the query is processed without any exceptions
- Predicates for each supplied parameter are combined using with an implicit LOGICAL AND if matchOnAnyParameter is unspecified or false. If it is specified as true then predicates for each supplied parameters are combined using a LOGICAL OR

3.9 Canonical Query: FindAssociatedObjects

The canonical query FindAssociatedObjects query allows clients to find RegistryObjects that are associated with the specified RegistryObject and matched the specified criteria.

3.9.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
associationType	Matches associated RegistryObjects where the Association's associationsType, rim:/Association/@associationType references a RegistryObject whose objectType attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value	string		0..1
matchOnAnyParameter	If true then use logical OR between predicates for each parameter	boolean	false	0..1
sourceObjectId	Matches target RegistryObjects of Associations where the source RegistryObject's id matches rim:Association/@sourceObject. Allows use of "%" wildcard character to match multiple characters. Allows use of "?" wildcard character to match a single character.	string		0..1
sourceObjectType	Matches target RegistryObjects of Associations whose sourceObject attribute references a RegistryObject whose objectType attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value	string		0..1
targetObjectId	Matches source RegistryObjects of Associations where the target RegistryObject's id matches rim:Association/@targetObject. Allows use of "%" wildcard character to match multiple characters. Allows use of "?" wildcard character to match a single character.	string		0..1
targetObjectType	Matches source RegistryObjects of Associations whose targetObject attribute	string		0..1

	references a RegistryObject whose objectType attribute matches the id of the ClassificationNode where rim:ClassificationNode/@path matches specified value			

3.9.2 Query Semantics

- All parameters are optional
- The server MUST return the objects matching the query if the query is processed without any exceptions
- Either sourceObjectId or targetObjectId MUST be specified. If niether are specified then InvalidQueryException fault MUST be returned
- Both sourceObjectId and targetObjectId MUST NOT be specified. If both are specified then InvalidQueryException fault MUST be returned
- Predicates for each supplied parameter are combined using with an implicit LOGICAL AND if matchOnAnyParameter is unspecified or false. If it is specified as true then predicates for each supplied parameters are combined using a LOGICAL OR

3.10 Canonical Query: GetClassificationSchemesById

The canonical query GetClassificationSchemesById allows clients to fetch specified ClassificationSchemes.

3.10.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
id	Matches rim:ClassificationScheme/@id. Allows use of "%" wildcard character to match multiple characters. Allows use of "?" wildcard character to match a single character.	string		0..1

3.10.2 Query Semantics

- The server MUST return the objects matching the query if the query is processed without any exceptions
- The depth parameter of the QueryRequest may be used to pre-fetch the ClassificationNodes of matches ClassificationSchemes

3.11 Canonical Query: GetRegistryPackagesByMemberId

The canonical query GetRegistryPackagesByMemberId allows clients to get the RegistryPackages that a specified RegistryObject is a member of.

3.11.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
memberId	Matches RegistryPackages that have a RegistryObject as member where the RegistryObject's id rim:Registry/@id matches the specified value. Allows use of "%" wildcard character to match multiple characters. Allows use of "?" wildcard character to match a single character.	string		0..1

3.11.2 Query Semantics

- The server MUST return the objects matching the query if the query is processed without any exceptions

3.12 Canonical Query: GetMembersByRegistryPackageId

The canonical query GetMembersByRegistryPackageId allows clients to get the RegistryObjects that are members of a specified RegistryPackage.

3.12.1 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
packageId	Matches RegistryObjects that are members of a RegistryPackage whose id rim:RegistryPackage/@id matches the specified value. Allows use of "%" wildcard character to match multiple characters. Allows use of "?" wildcard character to match a single character.	string		0..1

3.12.2 Query Semantics

- The server MUST return the objects matching the query if the query is processed without any exceptions

3.13 Canonical Query: KeywordSearch

The canonical query KeyWordSearch allows clients to find RegistryObjects and RepositoryItems that contain text that matches keywords identified by specified search patterns.

3.13.1 Canonical Indexes

This query defines a set of canonical index names as defined by table below. Each index name is associated with a particular type of information that it indexes. A server MUST index all information that is defined by the canonical indexes below. A server MAY define additional indexes to index information not specified by this section.

Index Name	Description
name.localizedString.value	Indexes the value of all localized string in all Name elements of all RegistryObjects
description.localizedString.value	Indexes the value of all localized string in all Description elements of all RegistryObjects
slot.name	Indexes the name of all slots on all RegistryObjects
slot.valueList.value	Indexes the value of all slots on all RegistryObjects
repositoryItem	Indexes the text of all text based repositiry items associated with ExtrinsicObjects
personName.firstName	Indexes the firstName attribute of PersonName elements in all Person objects
personName.middleName	Indexes the middleName attribute of PersonName elements in all Person objects
personName.lastName	Indexes the lastName attribute of PersonName elements in all Person objects
emailAddress.address	Indexes the address attribute of all EmailAddress objects
postalAddress.city	Indexes the city attribute of all PostalAddress elements contained within any RegistryObject
postalAddress.country	Indexes the country attribute of all PostalAddress elements contained within any RegistryObject
postalAddress.postalCode	Indexes the postalCode attribute of all PostalAddress elements contained within any RegistryObject
postalAddress.stateOrProvince	Indexes the stateOrProvince attribute of all PostalAddress elements contained within any RegistryObject
postalAddress.street	Indexes the street attribute of all PostalAddress elements contained within any RegistryObject

3.13.2 Parameter Summary

Parameter	Description	Data Type	Default Value	Cardinality
keywords	A space separated list of keywords to search for	string		1

3.13.3 Query Semantics

The value of the keywords parameter may consist of multiple terms where each term is separated by one or more spaces

Example: ebxml regrep
Semantics: Matches objects containing either “ebxml” or “regrep”

- A term may be enclosed in double-quotes to include white space characters as a literal value

  Example: “ebxml regrep”
  Semantics: Matches objects containing “ebxml regrep”

- Terms may be specified using wildcard characters where '*' matches one or more characters and “?” matches a single character (Issue: what about inconsistency with “%”??)

  Example: eb?ml reg*

- Terms may be combined using boolean operators “AND”, “OR” and “NOT”. Absence of a boolean operator between terms implies an implicit OR operator between them

  Example: ebxml AND regrep
  Semantics: Matches objects containing “ebxml” and “regrep”

  Example: ebxml NOT regrep
  Semantics: Matches objects containing “ebxml” and not containg “regrep”

  Example: ebxml OR regrep
  Semantics: Matches objects containing “ebxml” or “regrep”

  Example: ebxml regrep
  Semantics: Matches objects containing “ebxml” or “regrep”

- Terms may be grouped together using “(“ at the beging and “)” at the end of the group. Grouping allowing boolean operators to be applied to a group of terms as a whole and enables more flexible searches

Example: ebxml AND (registry OR regrep)
Semantics: Matches objects containing both “ebxml” and either “registry” or “regrep”

- The server MUST return all RegistryObjects that contain indexed data matching the semantics of the keywords parameter
- The server MUST return all ExtrinsicObjects that have a repository item that contains indexed data matching the semantics of the keywords parameter

4 LifecycleManager Interface

The LifecycleManager interface allows a client to perform various lifecycle management operations on RegistryObjects. These operations include submitting RegistryObjects to the server, updating RegistryObjects in the server, creating new versions of RegistryObjects in the server and removing RegistryObjects from the server.

A server MUST implement the LifecucleManager interface as an endpoint.

This specification does not specify explicit workflow support beyond basic CRUD operations described here. Such workflow may be implemented using status attribute in combination with subscription and notification feature.

4.1 SubmitObjects Protocol

The SubmitObjects protocol allows a client to submit RegistryObjects to the server. It also allows a client to completely replace existing RegistryObjects in the server.

A client initiates the SubmitObjects protocol by sending an SubmitObjectsRequest message to the LifecycleManager endpoint.

The LifecycleManager sends an RegistryResponse back to the client as response.

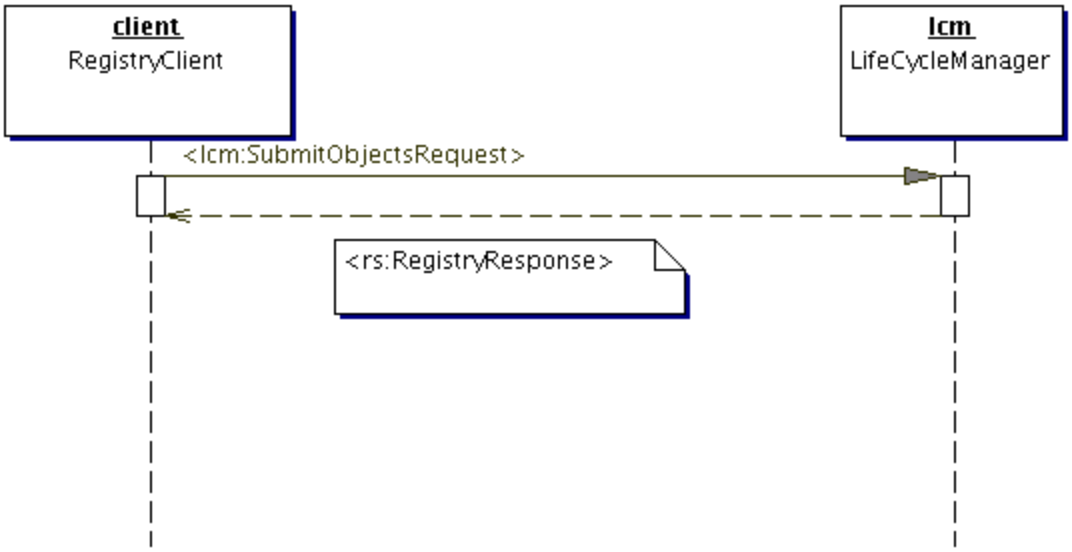

Illustration 2: SubmitObjects Protocol (replace??)

4.1.1 SubmitObjectsRequest

The SubmitObjectsRequest message is sent by a client to submit RegistryObjects to the server.

The server MUST apply any configured validation services for RegistryObjects in a SubmitObjectsRequest before committing the request as described here.

The server MUST apply any configured cataloging services for RegistryObjects in a SubmitObjectsRequest before committing the request as described here.

4.1.1.1 Syntax

```
<element name="SubmitObjectsRequest">
  <annotation>
    <documentation xml:lang="en">This SubmitObjects protocol allows a client to
      submit RegistryObjects to the server. It also allows a client to completely
      replace existing RegistryObjects in the server.</documentation>
  </annotation>
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <element ref="rim:RegistryObjectList" minOccurs="0" maxOccurs="1"/>
        </sequence>
        <attribute name="mode" use="optional" default="createOrReplace">
          <simpleType>
            <restriction base="NCName">
              <enumeration value="createOrReplace"/>
              <enumeration value="createOnly"/>
              <enumeration value="replaceOnly"/>
            </restriction>
          </simpleType>
        </attribute>
      </extension>
    </complexContent>
  </complexType>
</element>
```

4.1.1.2 Description

- Element RegistryObjectList - This element specifies a set of RegistryObject instances that are being submitted to the registry. The RegistryObjects in the list may be brand new objects being submitted to the registry or they may be current objects already existing in the registry.
- Attribute mode – Do we need it, why not use just the default behavior?? This attribute specifies the semantics for how the server should handle RegistryObjects being submitted when they already exist in the server:
  - createOrReplace (default) - If an object does not exist, server MUST create it as a new object. If an object already exists, server must replace the existing object with the submitted object
  - createOnly - If an object already exists, the server must return an Exception fault message
  - updateOnly - If an object does not exists, the server must return an Exception fault message

4.1.1.3 Returns

This request returns a RegistryResponse.

4.1.1.4 Exceptions

No additional exceptions are defined beyond the exceptions common (where best to define these??) to all requests.

4.1.2 RegistryResponse

TODO??

4.1.3 Unique ID Generation

Following needs to be updated for RegRep4??

A Submitter MUST supply the id attribute for submitted objects. If the id is not specified then the registry MUST return an InvalidRequestException.

If the id and lid match the id and lid of an existing RegistryObject within the home registry, then the registry MUST treat it as an Update action upon the existing RegistryObject.

If the id matches the id of an existing RegistryObject within the home registry but the lid does not match that existing object's lid, then the registry MUST return an InvalidRequestException.

If the lid matches the lid of an existing RegistryObject within the home registry but the id does not match that existing object's id, then the registry MUST create the newly submitted object as a new version of the existing object.

If the Submitter supplies the id and it is a valid URN then the registry MUST honor the Submitter-supplied id value and use it as the value of the id attribute of the object in the registry. If the id is not a valid URN then the registry MUST treat it as a temporary id and replace it, and all references to it within the request, with a registry generated universally unique id. A registry generated universally unique id value MUST conform to the format of a URN that specifies a DCE 128 bit UUID as specified in [UUID]:

(e.g. *urn:uuid:a2345678-1234-1234-123456789012*)

4.1.4 ID Attribute And Object References

Following needs to be updated for RegRep4??

The id attribute of an object MAY be used by other objects to reference that object. Within a SubmitObjectsRequest, the id attribute MAY be used to refer to an object within the same SubmitObjectsRequest as well as to refer to an object within the registry. An object in the SubmitObjectsRequest that needs to be referred to within the request document MAY be assigned an id by the submitter so that it can be referenced within the request. The submitter MAY give the object a valid URN, in which case the id is permanently assigned to the object within the registry. Alternatively, the submitter MAY assign an arbitrary id that is not a valid URN as long as the id is a unique anyURI value within the request document. In this case the id serves as a linkage mechanism within the request document but MUST be replaced with a registry generated id upon submission.

When an object in a SubmitObjectsRequest needs to reference an object that is already in the registry, the request MAY contain an ObjectRef whose id attribute is the id of the object in the registry. This id is by definition a valid URN. An ObjectRef MAY be viewed as a proxy within the request for an object that is in the registry.

4.1.5 Audit Trail Requirements

A server MUST create AuditableEvents *after* successfully processing a SubmitObjectsRequest and as part of the same transaction as the request.

The server MUST create a single AuditableEvent object with eventType *Created* for all the RegistryObjects created during processing of a SubmitObjectsRequest.

The server MUST create a single AuditableEvent object with eventType *Replaced* *(new: need to add to schema??)* for all the RegistryObjects replaced during processing of a SubmitObjectsRequest.

4.1.6 Sample SubmitObjectsRequest

The following simplified example shows a SubmitObjectsRequest that submits a single Organization object to the registry.

```
<lcm:SubmitObjectsRequest>
  <rim:RegistryObjectList>
    <rim:RegistryObject xsi:type="rim:OrganizationType" lid="${LOGICAL_ID}"
     id="${ID}" ...>

     ...

    </rim:RegistryObject>
  </rim:RegistryObjectList>
</SubmitObjectsRequest>
```

4.2 The Update Objects Protocol

The UpdateObjectsRequest protocol allows a client to make partial updates to one or more RegistryObjects that already exist in the server. This protocol allows *partial* update of RegistryObjects rather than a *complete replacement*. A client should use the SubmitObjects protocol for complete replacement of RegistryObjects.

A server MUST return <rs:InvalidRequestException> fault message if the client attempts to update the id, lid or objectType attribute of a RegistryObject.

Issue: How to access control who can change status attribute??

Full object replacement is an error??

Document scope is RegistryObject??

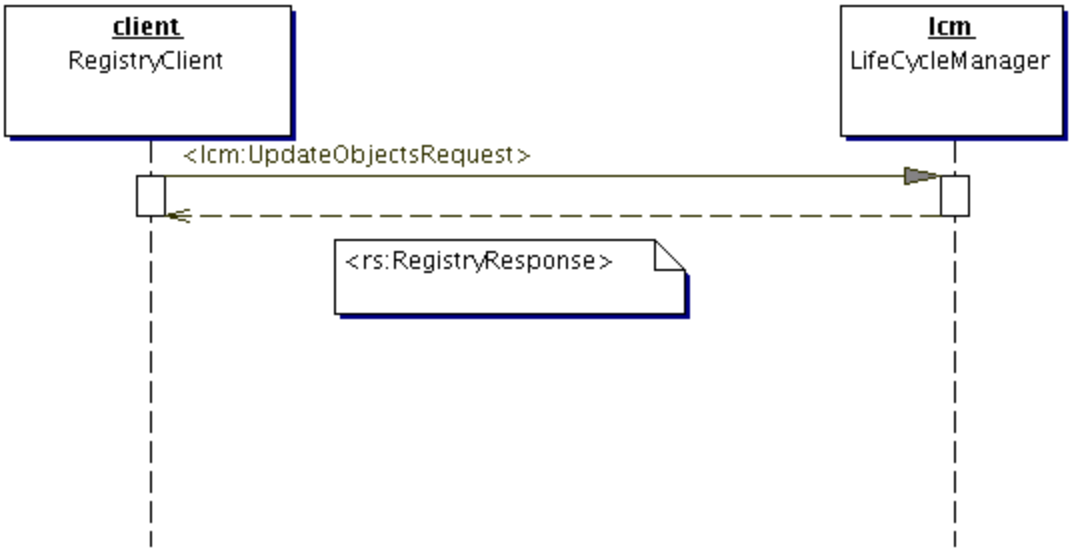

Illustration 3: UpdateObjects Protocol (replace??)

4.2.1 UpdateObjectsRequest

The UpdateObjectsRequest message is sent by a client to partially update existing RegistryObjects in the server. An UpdateObjectsRequest identifies a set of RegistryObjects as target objects to be updated by the request. It also specifies the details of the update action that modifies each target object. Update actions may insert a fragment within a target object, delete an existing fragment from a target object or update an existing fragment within the target object.

The server MUST apply any configured validation services for RegistryObjects in a SubmitObjectsRequest before committing the request as described here.

The server MUST apply any configured cataloging services for RegistryObjects in a SubmitObjectsRequest before committing the request as described here.

4.2.1.1 Syntax

```
<element name="UpdateObjectsRequest">
  <annotation>
    <documentation xml:lang="en">This UpdateObjectsRequest allows a client to
      updates elements and attributes within RegistryObjects already existing in the
      server.</documentation>
  </annotation>
  <complexType>
    <complexContent>
      <extension base="rs:RegistryRequestType">
        <sequence>
          <!-- QueryInvocation and ObjectRefList select objects to update -->
          <element ref="rim:QueryInvocation" minOccurs="0" maxOccurs="1" />
          <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />

          <!-- Specifies how to update selected objects -->
          <element name="UpdateAction" type="tns:UpdateActionType" minOccurs="1"
      maxOccurs="unbounded"/>
```

```
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</element>
```

4.2.1.2 Description

- Element QueryInvocation - Specifies a query to be invoked. A server MUST use all objects that match the specified query in addition to any other objects identified by the ObjectRefList element as targets of the update action.
- Element ObjectRefList - Specifies a collection of references to existing RegistryObject instances in the registry. A server MUST use all objects that are referenced by this element in addition to any other objects identified by the QueryInvocation element as targets of the update action.
- Element UpdateAction – Specifies the details of how to update the target objects.

4.2.1.3 Returns

This request returns a RegistryResponse.

4.2.1.4 Exceptions

No additional exceptions are defined beyond the exceptions common (where best to define these??) to all requests.

4.2.2 UpdateAction

An UpdateRequest contains one or more UpdateActions. Each UpdateObjectsRequest defines a specific update action to be performed on each target object.

4.2.2.1 Syntax

```
  <complexType name="UpdateActionType">
    <annotation>
      <documentation xml:lang="en">
      </documentation>
    </annotation>
    <sequence>
      <!-- Value for attribute or element -->
      <element name="Value" type="rim:ValueType" minOccurs="1" maxOccurs="1"/>
      <!--
      Value of selector is an XPATH expression that uniquely identifies an
attribute
      or an element within target documents.
      -->
      <element name="Selector" type="rim:QueryExpressionType" minOccurs="1"
maxOccurs="1"/>
    </sequence>

    <!--
    Specifies whether to insert, update or delete a fragment from target
document.
```

```
    -->
    <attribute name="mode">
      <simpleType>
        <restriction base="NCName">
          <enumeration value="insert"/>
          <enumeration value="update"/>
          <enumeration value="delete"/>
        </restriction>
      </simpleType>
    </attribute>
  </complexType>
```

4.2.2.2 Description

Element Selector – Is a QueryExpressionType that contains the expression that identifies a fragment of the resource representation to be updated.

The value of this element MUST conform to the queryLanguage specified in the queryLanguage attribute of the Selector. A resource MUST generate an InvalidExpressionException fault if the expression is invalid. If the expression syntax is not valid with respect to the queryLanguage then a resource SHOULD specify a fault detail of "InvalidExpressionSyntaxException". If the expression value is not valid for the resource type then the resource SHOULD specify a fault detail of "InvalidExpressionValueException".

A server MUST minimally support XPATH 1.0 as the queryLanguage for Selector element.

- Element Value - This element contains the value to be written to the target object. If the mode attribute is "Insert" or "Update" then this element MUST be present. If the mode is "Delete" then this element MUST NOT be present. A resource MUST generate an InvalidRequestException fault if it receives a message with a Value cardinality that is not valid for the Mode attribute.
- Attribute mode – This attribute specifies the semantics for how the server should update target objects:
  - insert - Indicates that the Value MUST be added to the target object. If the selector targets a repeated element (maxOccurs > 1), the fragment MUST be added at the end. If the selector targets a non-repeated element (maxOccurrs = 1) that already exists, the resource MUST generate a FragmentAlreadyExistsException fault. If the selector targets an existing item of a repeated element, the Value MUST be added before the existing item.
  - update – Indicates that the fragment identified by selector MUST be replaced by the specified Value in its place. If the selector resolves to nothing then this fragment element does not result in any change to the target object.
  - delete - indicates that the fragment identified by selector MUST be deleted from the target object if it is present.

4.2.3 Audit Trail Requirements

A server MUST create AuditableEvents *after* successfully processing a UpdateObjectsRequest and as part of the same transaction as the request.

The server MUST create a single AuditableEvent object with eventType *Updated* for all the RegistryObjects updated during processing of a SubmitObjectsRequest.

4.2.4 Sample UpdateObjectsRequest

```
TBD??
```

4.3 RemoveObjects Protocol

The Remove Objects protocol allows a client to remove or delete one or more RegistryObject instances from the server.

A client initiates the RemoveObjects protocol by sending an RemoveObjectsRequest message to the LifecycleManager endpoint.

The LifecycleManager sends an RegistryResponse back to the client as response.

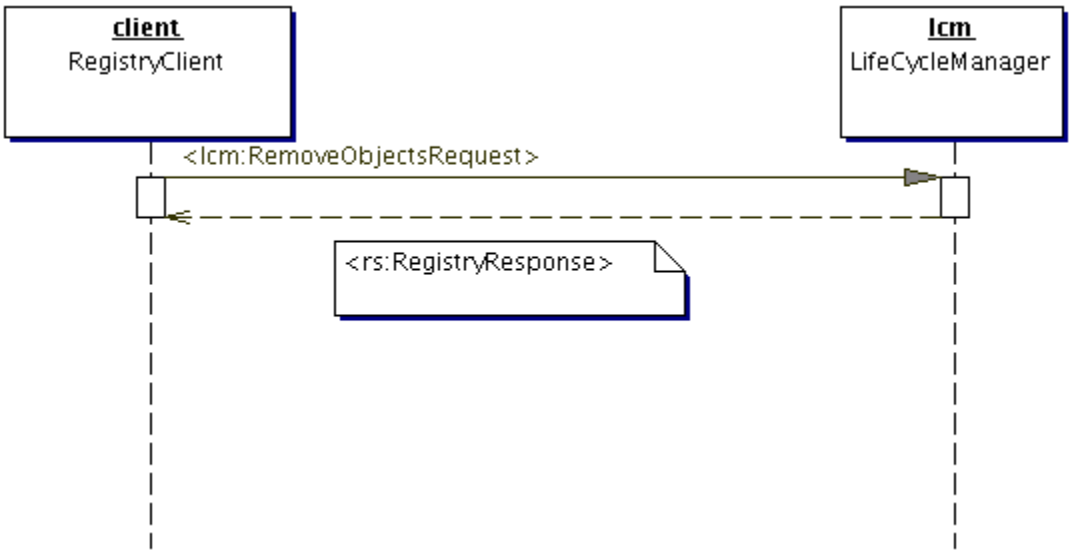

Illustration 4: RemoveObjects Protocol (replace??)

4.3.1 RemoveObjectsRequest

The RemoveObjectsRequest message is sent by a client to remove one or more existing RegistryObjects from the server.

4.3.1.1 Syntax

```
<element name="RemoveObjectsRequest">
  <annotation>
    <documentation xml:lang="en">
      The ObjectRefList is the list of
      refs to the registry entrys being removed
    </documentation>
  </annotation>
  <complexType>
```

```
      <complexContent>
        <extension base="rs:RegistryRequestType">
          <sequence>
            <element ref="rim:QueryInvocation" minOccurs="0" maxOccurs="1" />
            <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />
          </sequence>
          <attribute name="deletionScope" default="urn:oasis:names:tc:ebxml-
regrep:DeletionScopeType:DeleteAll" type="rim:referenceURI" use="optional"/>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

4.3.1.2 Description

- Attribute deletionScope - This attribute specifies the scope of impact of the RemoveObjectsRequest. The value of the deletionScope attribute MUST be a reference to a ClassificationNode within the canonical DeletionScopeType ClassificationScheme as described in ebRIM. A server MUST support the deletionScope types as defined by the canonical DeletionScopeType ClassificationScheme. The canonical DeletionScopeType ClassificationScheme may easily be extended by adding additional ClassificationNodes to it.

  The following canonical ClassificationNodes are defined for the DeletionScopeType ClassificationScheme:

  - DeleteRepositoryItemOnly - Specifies that the registry MUST delete the RepositoryItem for the specified ExtrinsicObjects but MUST NOT delete the specified ExtrinsicObjects
  - DeleteAll (default) - Specifies that the request MUST delete both the RegistryObject and the RepositoryItem (if any) for the specified objects
- Element QueryInvocation - Specifies a query to be invoked. A server MUST remove all objects that match the specified query in addition to any other objects identified by the ObjectRefList element.
- Element ObjectRefList - *S*pecifies a collection of references to existing RegistryObject instances in the registry. A server MUST remove all objects that are referenced by this element in addition to any other objects identified by the QueryInvocation element.

Issue: Should we not do referential integrity checks and allow dangling refs to achieve flexibility and performance?? Should we add a canonical query FindDanglingReferences(type)??

4.3.1.3 Returns:

This request returns a RegistryResponse.

4.3.1.4 Exceptions:

In addition to the exceptions common to all requests, the following exceptions MAY be returned:

- UnresolvedReferenceException - Indicates that the requestor referenced an object within the request that was not resolved during the processing of the request.

- ReferencesExistException - Indicates that the requestor attempted to remove a RegistryObject while references to it still exist. Note that it is valid to remove a RegistryObject and all RegistryObjects that refer to it within the same request. In such cases the ReferencesExistException MUST not be thrown.

4.3.2 Audit Trail Requirements

A server MUST create AuditableEvents *after* successfully processing a RemoveObjectsRequest and as part of the same transaction as the request.

The server MUST create a single AuditableEvent object with eventType *Updated* for all the RegistryObjects updated during processing of a SubmitObjectsRequest.

4.3.3 Sample UpdateObjectsRequest

```
TBD??
```

5 Validator Interface

The Validator interface allows a client to validate objects already in the server. A server MUST implement the Validator interface as an endpoint. The Validator interface validates objects using Validator Plugins specific to the type of object being validated.

5.1 ValidateObjects Protocol

A client validates RegistryObjects residing in the server using the *ValidateObjects* protocol supported by the validateObjects operation of the Validator interface.

A client initiates the ValidateObjects protocol by sending an ValidateObjectsRequest message to the Validator endpoint.

The Validator endpoint sends an ValidateObjectsResponse back to the client as response. The ValidateObjectsResponse contains information on whether the objects were valid and if invalid objects were found it includes any validation errors that were encountered.

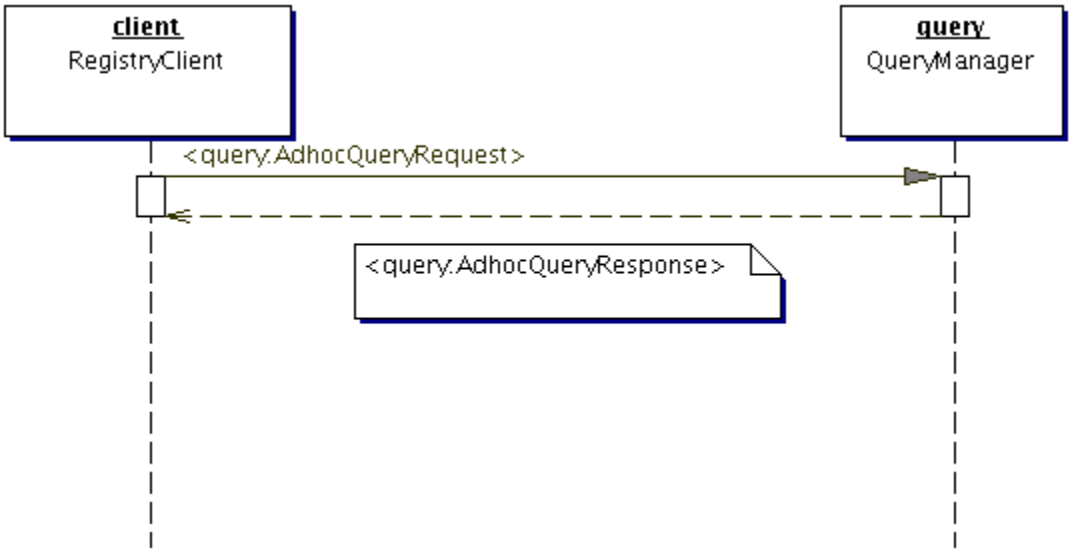

Illustration 5: ValidateObjects Protocol *Need to update figure??*

5.1.1 ValidateObjectsRequest

The ValidateObjectsRequest message is sent by client to the Validator interface to validate objects that are already resident in the server.

5.1.1.1 Syntax

```
<element name="ValidateObjectsRequest">
```



```
    <annotation>
      <documentation xml:lang="en">Request to validate specified
objects.</documentation>
    </annotation>
    <complexType>
      <complexContent>
        <extension base="rs:RegistryRequestType">
          <sequence>
            <!-- QueryInvocation and ObjectRefList select objects to validate
-->
            <element ref="rim:QueryInvocation" minOccurs="0" maxOccurs="1" />
            <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />
            <element name="OriginalObjects" type="rim:RegistryObjectListType"/
>
            <!-- The Invocation Control File (optional). -->
            <element maxOccurs="unbounded" minOccurs="0"
name="InvocationControlFile" type="rim:ExtrinsicObjectType"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

5.1.1.2 Example

TODO??

5.1.1.3 Description

Issue: Editorial - Need to list items in order of schema, schema should be in alpha order??

- Element QueryInvocation - Specifies a query to be invoked. A server MUST validate all objects that match the specified query.
- Element ObjectRefList - Specifies a collection of references to existing RegistryObject instances in the registry. A server MUST validate all objects that are referenced by this element.
- Element OriginalObjects - Specifies a collection of RegistryObject instances. A server MUST validate all objects that are contained in this element.
- Element InvocationControlFile – Specifies an ExtrinsicObject that is used to control the validation process in a type specific manner. See Canonical XML Validator plugin for an example.

5.1.1.4 Response

This request returns ValidateObjectsResponse as response.

5.1.1.5 Exceptions

Issue: Need to define more fine-grained exception hierarchy??

In addition to the exceptions common to all requests defined in ??, the following exceptions MAY be returned:

- ValidationException: signifies that an exception was encountered during the validateObjects operation

5.1.2 ValidateObjectsResponse

Currently ValidateObjectsResponse is a simple extension to RegistryResponseType and does not define additional attributes or elements. Issue: How to return all ValidationExceptions in a single response??

6 Cataloger Interface

The Cataloger interface allows a client to catalog or index objects already in the server. A server MUST implement the Cataloger interface as an endpoint. The Cataloger interface catalogs objects using Cataloger Plugins specific to the type of object being validated.

6.1 CatalogObjects Protocol

A client catalogs RegistryObjects residing in the server using the *CatalogObjects* protocol supported by the catalogObjects operation of the Cataloger interface.

A client initiates the CatalogObjects protocol by sending an CatalogObjectsRequest message to the Cataloger endpoint.

The Cataloger endpoint sends a CatalogObjectsResponse back to the client as response.

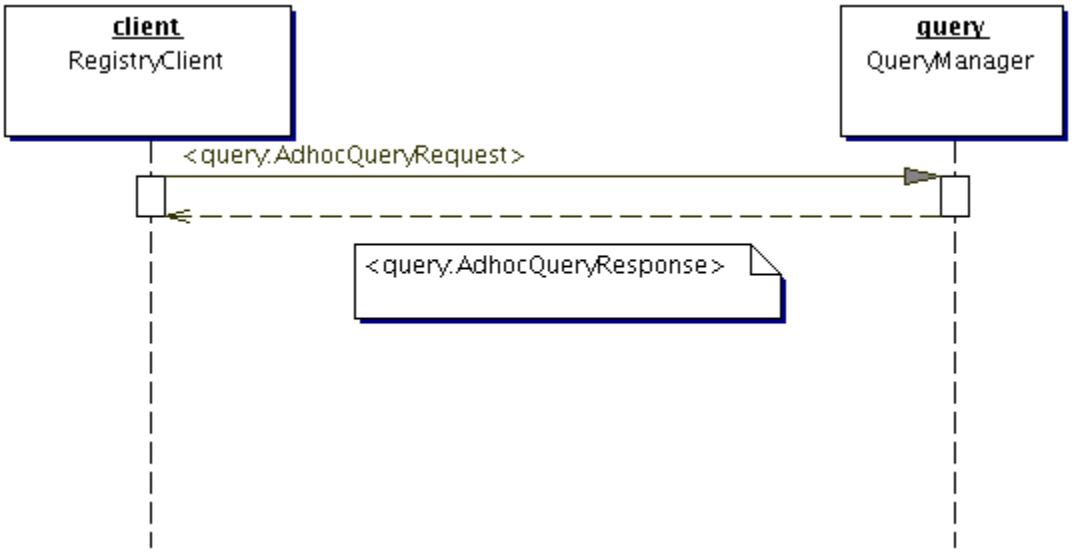

Illustration 6: CatalogObjects Protocol Need to update figure??

6.1.1 CatalogObjectsRequest

The CatalogObjectsRequest message is sent by client to the Cataloger interface to catalog objects that are already resident in the server.

6.1.1.1 Syntax

```
<element name="CatalogObjectsRequest">
  <annotation>
```

```
      <documentation xml:lang="en">Request to catalog specified
objects.</documentation>
    </annotation>
    <complexType>
      <complexContent>
        <extension base="rs:RegistryRequestType">
          <sequence>
            <!-- QueryInvocation and ObjectRefList select objects to catalog
-->
            <element ref="rim:QueryInvocation" minOccurs="0" maxOccurs="1" />
            <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />
            <element name="OriginalObjects" type="rim:RegistryObjectListType"/
>
            <!-- The Invocation Control File (optional). -->
            <element maxOccurs="unbounded" minOccurs="0"
name="InvocationControlFile" type="rim:ExtrinsicObjectType"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

6.1.1.2 Example

TODO??

6.1.1.3 Description

- Element QueryInvocation - Specifies a query to be invoked. A server MUST catalog all objects that match the specified query.
- Element ObjectRefList - Specifies a collection of references to existing RegistryObject instances in the registry. A server MUST catalog all objects that are referenced by this element.
- Element OriginalObjects - Specifies a collection of RegistryObject instances. A server MUST catalog all objects that are contained in this element.
- Element InvocationControlFile – Specifies an ExtrinsicObject that is used to control the validation process in a type specific manner. See Canonical XML Catalogor plugin for an example.

6.1.1.4 Response

This request returns CatalogObjectsResponse as response.

6.1.1.5 Exceptions

In addition to the exceptions common to all requests defined in ??, the following exceptions MAY be returned:

- CatalogException: signifies that an exception was encountered during the catalogObjects operation

6.1.2 CatalogObjectsResponse

The CatalogObjectsResponse message is sent by the Cataloger endpoint in response to an CatalogObjectsRequest.

6.1.2.1 Syntax

```
  <element name="CatalogObjectsResponse">
    <annotation>
      <documentation xml:lang="en">Response to request to catalog specified
objects.</documentation>
    </annotation>
    <complexType>
      <complexContent>
        <extension base="rs:RegistryResponseType">
          <sequence>
            <element name="CatalogedObjects"
type="rim:RegistryObjectListType"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

6.1.2.2 Example

TODO??

6.1.2.3 Description

In addition to elements and attributes defined by RegistryResponseType the following are defined:

- Element CatalogedObjects – Contains the RegistryObjects that are produced as output of the catalogObjects operation. Typically this list contains the objects that were input to the catalogObjects operation, as well as new objects that were the output of the catalogObjects operation. The input objects MAY be modified by the cataloger as a result of the catalogObjects operation.

7 Server Plugin SPI

Deployments of the server MAY extend the core functionality of the server by using function-specific software modules called plugins. A plugin extends the server by adding additional functionality to it. A plugin MUST conform to standard interfaces called Service Provider Interfaces (SPI) as defined by this specification.

This chapter specifies the Service Provider Interfaces (SPI) that defines the standard interface for various types of server plugins. The interfaces are described in form of WSDL specification. A server may implement these interfaces as external web services invoked by the server using SOAP or as plugin modules that share the same process as the server and are invoked by local function calls.

Examples of types of server plugins include, but are not limited to query plugin, validator plugin and cataloger plugin.

This specification does not define how a plugin is configured within a server. Nor does it define whether or how, plugin configuration functionality is made available by the server to clients.

7.1 Query Plugins

Query plugins allow a server to implement support for a parameterized query as a plugin. Since query plugins are software modules, they able to handle highly specialized query semantics that cannot be expressed in most query languages. A specific instance of a query plugin is designed and configured to handle a specific parameterized query.

7.1.1 Query Plugin Interface

A Query plugin implements the QueryManager interface. A QueryManager endpoint MUST delegate an executeQuery operation to a Query plugin if a Query plugin has been configured for the requested parameterized query. A Query plugin MUST process the query and return a QueryResponse or fault message to the QueryManager. The QueryManager MUST then deliver that response to the client.

7.2 Validator Plugins

Validator plugins allow a server to validate objects being submitted during the processing of a SubmitObjectsRequest or being validate during the processing of a ValidateObjectsRequest.

A specific instance of a Validator plugin is designed and configured to validate a specific type of object. For example, The canonical XML Validator plugin is designed and configured to validate XML Objects using Schematron documents as InvocationControlFile.

7.2.1 Validator Plugin Interface

A Validator plugin implements the Validator interface. The server's Validator endpoint MUST delegate a validateObjects operation to any number of Validator plugins using the following algorithm:

- The server selects the RegistryObjects that are the target of the validateObjects operations using the <rim:QueryInvocation> and <rim:ObjectRefList> elements. Any objects specified by the OriginalObjects element MUST be ignored by the server.

- The server partitions the set of target objects into multiple sets based upon the objectType attribute value for the target objects
- The server determines whether there is a Validator plugin configured for each objectType for which there is a set of target objects
- For each set of target objects that share a common objectType and for which there is a configured Validator plugin, the server MUST invoke the Validator plugin. The Validator plugin invocation MUST specify the target objects for that set using the OriginalObjects element. The server MUST NOT specify <rim:QueryInvocation> and <rim:ObjectRefList> elements when invoking validateObjects operation on a Validator plugin
- Each Validator plugin MUST process the ValidateObjectsRquest and return a ValidateObjectsResponse or fault message to the server's Validator endpoint.
- The server's Validator endpoint MUST then combine the results of the individual ValidateObjectsRequest to Validator plugins into a single unified ValidateObjectsResponse and return it to the client.

7.2.2 Canonical XML Validator Plugin

The canonical XML Validator plugin is a validator plugin that validates XML content using a Schematron file as InvocationControlFile. The Schematron file specifies validation rules using [Schematron] language to validate XML content. The server may configure the canonical XML Validator plugin such that it is invoked with an appropriate schematron file as InvocationControlFile based upon the objectType of the object being validated.

A server MUST implement the canonical XML Validator plugin.

7.3 Cataloger Plugins

Cataloger plugins allow a server to catalog objects being submitted during the processing of a SubmitObjectsRequest or being cataloged during the processing of a CatalogObjectsRequest.

A specific instance of a Cataloger plugin is designed and configured to catalog a specific type of object. For example, The canonical XML Cataloger plugin is designed and configured to catalog XML Objects using XSLT documents as InvocationControlFile.

7.3.1 Cataloger Plugin Interface

A Cataloger plugin implements the Cataloger interface. The server's Cataloger endpoint MUST delegate a catalogObjects operation to any number of Cataloger plugins using the following algorithm:

- The server selects the RegistryObjects that are the target of the catalogObjects operations using the <rim:QueryInvocation> and <rim:ObjectRefList> elements. Any objects specified by the OriginalObjects element MUST be ignored by the server.
- The server partitions the set of target objects into multiple sets based upon the objectType attribute value for the target objects
- The server determines whether there is a Cataloger plugin configured for each objectType for which there is a set of target objects

- For each set of target objects that share a common objectType and for which there is a configured Cataloger plugin, the server MUST invoke the Cataloger plugin. The Cataloger plugin invocation MUST specify the target objects for that set using the OriginalObjects element. The server MUST NOT specify <rim:QueryInvocation> and <rim:ObjectRefList> elements when invoking catalogObjects operation on a Cataloger plugin

- Each Cataloger plugin MUST process the CatalogObjectsRquest and return a CatalogObjectsResponse or fault message to the server's Cataloger endpoint.

- The server's Cataloger endpoint MUST then combine the results of the individual CatalogObjectsRequest to Cataloger plugins and commit these objects as part of the transaction associated with the request. It MUST then combine the individual CatalogObjectsResponse messages into a single unified CatalogObjectsResponse and return it to the client.

7.3.2 Canonical XML Cataloger Plugin

The canonical XML Cataloger plugin is a Cataloger plugin that catalogs XML content using an XSLT file as InvocationControlFile. The XSLT file specifies transformations rules using [XSLT] language to catalog XML content. The server may configure the canonical XML Cataloger plugin such that it is invoked with an appropriate XSLT file as InvocationControlFile based upon the objectType of the object being validated.

A server MUST implement the canonical XML Cataloger plugin.

8 Subscription and Notification

Subscription and Notification is a normative though optional feature.

A client MAY subscribe to events that transpire in the server by creating a Subscription. A server supporting Subscription and Notification feature MUST deliver a Notification to the subscriber when an event transpires that matches the event selection criteria specified by the client.

8.1 Server Events

Activities within the server result in events. [ebRIM] defines the AuditableEvent class, instances of which represent server events. Typically, a server creates AuditableEvent instances during the processing of client requests.

8.1.1 Pruning of Events

A server MAY periodically prune AuditableEvents in order to manage its resources. It is up to the server when such pruning occurs. It is up to the server to determine when undelivered events are purged. A server SHOULD perform such pruning by removing the older AuditableEvents first. What about CREATED events which establish owner for RegistryObject??

8.2 Notifications

A Notification message is used by the server to notify clients of events they have subscribed to. A Notification contains the RegistryObjects, or references to the RegistryObjects, that are affected by the event for which the Notification is being sent

Details for the Notification element are defined in [ebRIM].

8.3 Creating a Subscription

A client MAY create a subscription within a server if it wishes the server to send it a Notification when a specific type of event transpires. A client creates a subscription by submitting a <rim:Subscription> instance to the server using the standard SubmitObjects protocol. If a Subscription is submitted to the server that does not support event notification then the server MUST return an UnsupportedCapabilityException.

Submission of a Subscription object follows the same rules as submission of any other RegistryObject. Details for the Subscription element are defined in [ebRIM].

8.3.1 Subscription Authorization

A deployment MAY use custom Access Control Policies to decide which users are authorized to create a subscription and to what events. A server MUST return an AuthorizationException in the event that an unauthorized user submits a Subscription to a server.

8.3.2 Subscription Quotas

A server MAY use server specific policies to decide an upper limit on the number of Subscriptions a user is allowed to create. A server MUST return a QuotaExceededException in the event that an authorized user submits more Subscriptions than allowed by their server-specific quota.

8.3.3 Subscription Expiration

Each subscription defines a startTime and and endTime attribute which determines the period within which a Subscription is active. Outside the bounds of the active period, a Subsription MAY exist in an expired state within the server. A server MAY remove an expired Subscription at any time.

A Registry MUST NOT deliver notifications for an event to an expired Subscriptions. An expired Subscription MAY be renewed by updating the startTime and / or endTime for the Subscription using the UpdateObjects protocol.

8.3.4 Event Selection

A client MUST specify a Selector element within the Subscription to specify its criteria for selecting events of interest. The Selector element is of type <rim:QueryInvocationType> and specifies an parameterized query to be invoked with specified query parameters.

A server MUST process AuditableEvents and determine which Subscriptions match the event using the following algorithm:

- TODO??

8.4 Event Delivery

A client MUST specify a DeliveryInfo element within the Subscription to specify how the server should deliver events matching the subscription to the client.The DeliveryInfo element include a NotifyTo element which specifies an EndPoint Reference (EPR) as defined by [WS-Addressing]. The NotifyTo element contains a <wsa:Address> element which contains a URI to the endpoint.

Details for the DeliveryInfo element are defined in [ebRIM].

8.4.1 Delivery Mode

A client MAY specify a mode attribute in DeliveryInfo element of a Subscription. The mode attribute specifies how the client wishes to be notified of events. This specification defines the following modes and allows a server to support additional modes:

- urn:oasis:names:tc:ebxml-regrep:rim:DeliveryInfo:mode:push - This mode specifies that the client wishes the server to include complete RegistryObjects within the Notifications.
- urn:oasis:names:tc:ebxml-regrep:rim:DeliveryInfo:mode:push-pull - This mode specifies that the client wishes the server to include only ObjectRefs to RegistryObjects within the Notifications. The client MAY then “pull” the actual RegistryObjects from the server if needed using the Query protocol.

- urn:oasis:names:tc:ebxml-regrep:rim:DeliveryInfo:mode:pull - This mode specifies that the client wishes the server to not deliver any Notification to it. Instead the client MAY “pull” AuditableEvents of interest using the Query protocol.

8.4.2 Delivery to NotificationListener Web Service

If the URI in the <wsa:Address> element is a URL that uses the http protocol then the server MUST use this URL as the web service endpoint to deliver the Notification to. The target web service in this case MUST implement the NotificationListener interface.

8.4.3 Delivery to Email Address

If the URI in the <wsa:Address> element is a URL that uses the mailto protocol then the server MUST use this URL as the email address to deliver the Notification to via email. This specification does not define how a server is configured to send Notifications via email.

8.4.3.1 Processing Email Notification Via XSLT

A client MAY specify an XSLT style sheet within a DeliveryInfo element to process a Notification prior to it being delivered to an email address. The XSLT style sheet MAY be specified using a Slot in DeliveryInfo element where the Slot's name is “urn:oasis:names:tc:ebxml-regrep:rim:DeliveryInfo:email:xslt” and the Slots value is the id of an ExtrinsicObject whose repository item is the XSLT. The ExtrinsicObject and repository item MUST be submitted prior to or at the same time as the Subscription.

8.5 NotificationListener Interface

The NotificationListener interface allows a client to receive Notifications from the server for their Subscriptions. A client MUST implement the NotificationListener interface as an endpoint if they wish to receive Notifications via SOAP.

8.6 Notification Protocol

A server sends a Notification to a client using the *Notification* protocol supported by the onNotification operation of the NotificationListener interface.

A server initiates the Notification protocol by sending an Notification message to the NotificationListener endpoint registered within the Subscription for which the Notification is being delivered.

The onNotification operation does not send a response back to the server.

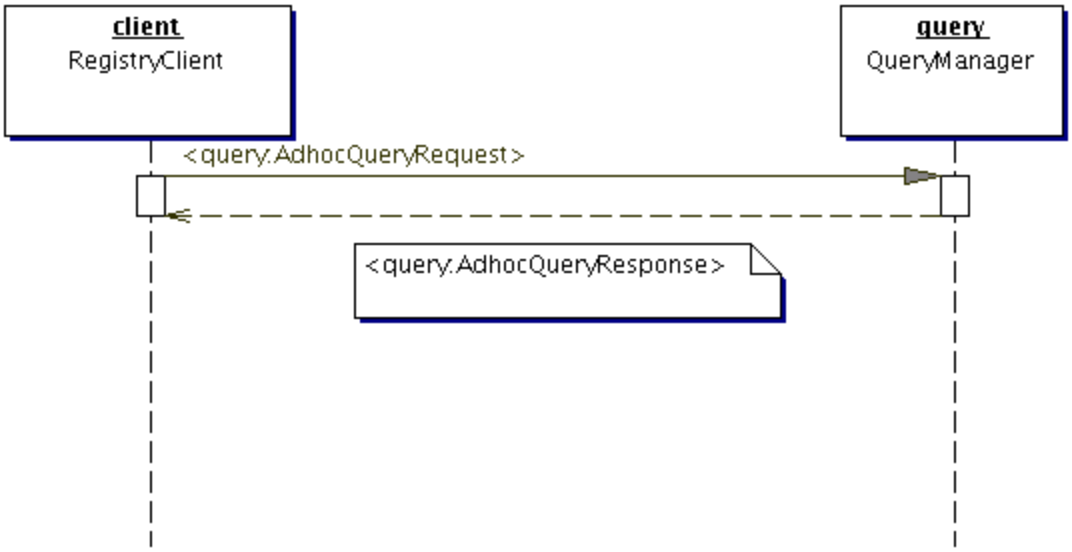

Illustration 7: CatalogObjects Protocol *Need to update figure??*

8.6.1 CatalogObjectsRequest

The CatalogObjectsRequest message is sent by client to the Cataloger interface to catalog objects that are already resident in the server.

8.6.1.1 Syntax

```
  <element name="CatalogObjectsRequest">
    <annotation>
      <documentation xml:lang="en">Request to catalog specified
objects.</documentation>
    </annotation>
    <complexType>
      <complexContent>
        <extension base="rs:RegistryRequestType">
          <sequence>
            <!-- QueryInvocation and ObjectRefList select objects to catalog
-->
            <element ref="rim:QueryInvocation" minOccurs="0" maxOccurs="1" />
            <element ref="rim:ObjectRefList" minOccurs="0" maxOccurs="1" />
            <element name="OriginalObjects" type="rim:RegistryObjectListType"/
>
            <!-- The Invocation Control File (optional). -->
            <element maxOccurs="unbounded" minOccurs="0"
name="InvocationControlFile" type="rim:ExtrinsicObjectType"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

8.6.1.2 Example

TODO??

8.6.1.3 Description

- Element QueryInvocation - Specifies a query to be invoked. A server MUST catalog all objects that match the specified query.
- Element ObjectRefList - Specifies a collection of references to existing RegistryObject instances in the registry. A server MUST catalog all objects that are referenced by this element.
- Element OriginalObjects - Specifies a collection of RegistryObject instances. A server MUST catalog all objects that are contained in this element.
- Element InvocationControlFile – Specifies an ExtrinsicObject that is used to control the validation process in a type specific manner. See Canonical XML Catalogor plugin for an example.

8.6.1.4 Response

This request returns CatalogObjectsResponse as response.

8.6.1.5 Exceptions

In addition to the exceptions common to all requests defined in ??, the following exceptions MAY be returned:

- CatalogException: signifies that an exception was encountered during the catalogObjects operation

8.6.2 CatalogObjectsResponse

The CatalogObjectsResponse message is sent by the Cataloger endpoint in response to an CatalogObjectsRequest.

8.6.2.1 Syntax

```
  <element name="CatalogObjectsResponse">
    <annotation>
      <documentation xml:lang="en">Response to request to catalog specified
objects.</documentation>
    </annotation>
    <complexType>
      <complexContent>
        <extension base="rs:RegistryResponseType">
          <sequence>
            <element name="CatalogedObjects"
type="rim:RegistryObjectListType"/>
          </sequence>
        </extension>
      </complexContent>
    </complexType>
  </element>
```

8.6.2.2 Example

TODO??

8.6.2.3 Description

In addition to elements and attributes defined by RegistryResponseType the following are defined:

- Element CatalogedObjects – Contains the RegistryObjects that are produced as output of the catalogObjects operation. Typically this list contains the objects that were input to the catalogObjects operation, as well as new objects that were the output of the catalogObjects operation. The input objects MAY be modified by the cataloger as a result of the catalogObjects operation.

8.7 Deleting a Subscription

A client MAY terminate a Subscription with a registry if it no longer wishes to be notified of events related to that Subscription. A client terminates a Subscription by deleting the corresponding Subscription object using the standard RemoveObjects protocol.

Removal of a Subscription object follows the same rules as removal of any other RegistryObject.

9 Cooperating Registries

Canonical objects should have a spec defined home attribute to a virtual “spec” RegRep instance.

Issue: Should we remove Federation Infomodel classes and instead rely on local queries against remoted registers?? Decisions here may impact Query Protocol??

10 Security Features

No longer defining registry as identity provider or authentication authority??.

Support pluggable authentication??

11 Native Language Support (NLS)

This chapter describes the Native Languages Support (NLS) features of ebXML Registry.

11.1 Terminology

The following terms are used in NLS.

NLS Term	Description
Coded Character Set (CCS)	CCS is a mapping from a set of abstract characters to a set of integers. [RFC 2130]. Examples of CCS are ISO-10646, US-ASCII, ISO-8859-1, and so on.
Character Encoding Scheme (CES)	CES is a mapping from a CCS (or several) to a set of octets. [RFC 2130]. Examples of CES are ISO-2022, UTF-8.
Character Set (charset)	• charset is a set of rules for mapping from a sequence of octets to a sequence of characters.[RFC 2277],[RFC 2278]. Examples of character set are ISO-2022-JP, EUC-KR. • A list of registered character sets can be found at [IANA].

11.2 NLS and Registry Protcol Messages

For the accurate processing of data in both registry client and registry services, it is essential for the recipient of a protocol message to know the character set being used by it.

A Registry Client SHOULD specify charset parameter in MIME header when they specify text/xml as Content-Type. A server MUST specify charset parameter in MIME header when they specify text/xml as Content-Type.

The following is an example of specifying the character set in the MIME header.

```
Content-Type: text/xml; charset=ISO-2022-JP
```

If a registry receives a protocol message with the charset parameter omitted then it MUST use the default charset value of "us-ascii" as defined in [RFC 3023].

Also, when an application/xml entity is used, the charset parameter is optional, and registry client and registry services MUST follow the requirements in Section 4.3.3 of [REC-XML] which directly address this contingency.

If another Content-Type is used, then usage of charset MUST follow [RFC 3023].

11.3 NLS Support in RegistryObjects

The information model XML Schema [RR-RIM-XSD] defines the <rim:InternationalStringType> for defining elements that contains a locale senstive string value.

```
  <complexType name="InternationalStringType">
    <sequence maxOccurs="unbounded" minOccurs="0">
      <element ref="tns:LocalizedString"/>
    </sequence>
  </complexType>
```

An InternationalStringType may contain zero or more LocalizedStrings within it where each LocalizedString contain a string value is a specified local language and character set.

```
<complexType name="LocalizedStringType">
  <attribute ref="xml:lang" default="en-US"/>
  <attribute default="UTF-8" name="charset"/>
  <attribute name="value" type="tns:FreeFormText" use="required"/>
</complexType>
```

Examples of such attributes are the “name” and “description” attributes of the RegistryObject class defined by [ebRIM] as shown below.

```
  <complexType name="InternationalStringType">
    <sequence maxOccurs="unbounded" minOccurs="0">
      <element ref="tns:LocalizedString"/>
    </sequence>
  </complexType>
  <element name="InternationalString" type="tns:InternationalStringType"/>
  <element name="Name" type="tns:InternationalStringType"/>
  <element name="Description" type="tns:InternationalStringType"/>

  <complexType name="LocalizedStringType">
    <attribute ref="xml:lang" default="en-US"/>
    <!--attribute name = "lang" default = "en-US" form = "qualified" type =
"language"/-->
    <attribute default="UTF-8" name="charset"/>
    <attribute name="value" type="tns:FreeFormText" use="required"/>
  </complexType>
  <element name="LocalizedString" type="tns:LocalizedStringType"/>
```

An element InternationalString is capable of supporting multiple locales within its collection of LocalizedStrings.

The above schema allows a single RegistryObject instance to include values for any NLS sensitive element in multiple locales.

The following example illustrates how a single RegistryObject can contain NLS sesnitive <rim:Name> and "<rim:Description> elements with their value specified in multiple locales. Note that the <rim:Name> and <rim:Description> use the <rim:InternationalStringType> as their type.

```
    <rim:ExtrinsicObject id="${ID}"  mimeType="text/xml">
      <rim:Name>
        <rim:LocalizedString xml:lang="en-US" value="customACP1.xml"/>
        <rim:LocalizedString xml:lang="fi-FI" value="customACP1.xml"/>
        <rim:LocalizedString xml:lang="pt-BR" value="customACP1.xml"/>
      </rim:Name>
      <rim:Description>
        <rim:LocalizedString xml:lang="en-US" value="A sample custom ACP"/>
        <rim:LocalizedString xml:lang="fi-FI" value="Esimerkki custom ACP"/>
        <rim:LocalizedString xml:lang="pt-BR" value="Exemplo de ACP
customizado
"/>
      </rim:Description>
    </rim:ExtrinsicObject>
```

Since locale information is specified at the sub-element level there is no language or character set associated with a specific RegistryObject instance.

11.3.1 Character Set of *LocalizedString*

The character set used by a locale specific String (LocalizedString) is defined by the charset attribute. Registry Clients SHOULD specify UTF-8 or UTF-16 as the value of the charset attribute of LocalizedStrings for maximum interoperability.

11.3.2 Language of *LocalizedString*

The language MAY be specified in xml:lang attribute (Section 2.12 [REC-XML]).

11.4 NLS and Repository Items

While a single instance of an ExtrinsicObject is capable of supporting multiple locales, it is always associated with a single repository item. The repository item MAY be in a single locale or MAY be in multiple locales. This specification does not specify any NLS requirements for repository items.

11.4.1 Character Set of Repository Items

When a submitter submits a repository item, they MAY specify the character set used by the respository item using the MIME *Content-Type* mime header for the mime multipart containing the repository item as shown below:

```
Content-Type: text/xml; charset="UTF-8"
```

Registry Clients SHOULD specify UTF-8 or UTF-16 as the value of the charset attribute of LocalizedStrings for maximum interoperability. A server MUST preserve the charset of a repository item as it is originally specified when it is submitted to the registry.

11.4.2 Language of Repository Items

The Content-language mime header for the mime bodypart containing the repository item MAY specify the language for a locale specific repository item. The value of the Content-language mime header property MUST conform to [RFC 1766].

This document currently specifies only the method of sending the information of character set and language, and how it is stored in a registry. However, the language information MAY be used as one of the query criteria, such as retrieving only DTD written in French. Furthermore, a language negotiation procedure, like registry client is asking a favorite language for messages from registry services, could be another functionality for the future revision of this document.

12 SOAP Binding

REST binding will be defined in a separate specification which will be based opun ATOM Syndication Format Atom Publishing Protocol and OpenSearch??.

Appendix A. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged

Participants:

- [Participant name, affiliation | Individual member]
- [Participant name, affiliation | Individual member]
- [Participant name, affiliation | Individual member]

Appendix B. Revision History

[optional; should not be included in OASIS standards]

Appendix C. Non-Normative Text